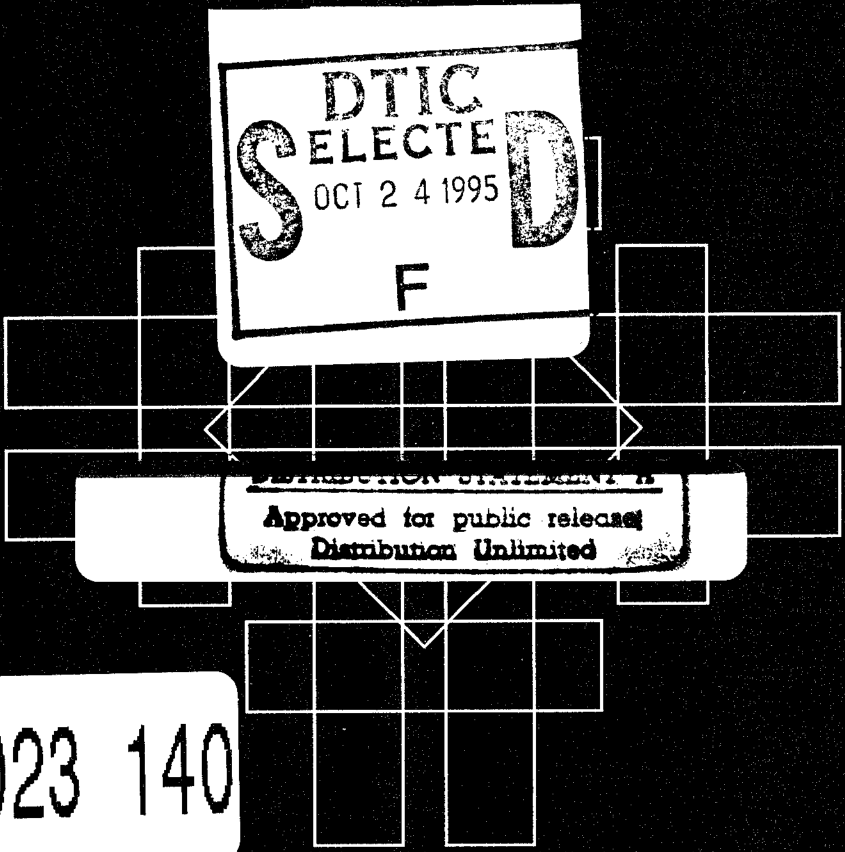
PROCEEDINGS OF THE TENTH ANNUAL SYMPOSIUM ON

COMPUTATIONAL GEOMETRY



Stony Brook, New York
June 6-8, 1994

Sponsored by the ACM Special Interest Groups

for Graphics and Algorithms and Computation Theory

acm PRESS

MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE
Jun 95

3. REPORT TYPE AND DATES COVERED
Final 9 May 94 - 8 May 95

4. TITLE AND SUBTITLE
Symposium on Computational Geometry

5. FUNDING NUMBERS
DAAH04-94-G-0225

6. AUTHOR(S)
Joseph Mitchell

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
State University of New York at Stony Brook
Stony Brook, NY 11794-3600

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
ARO 33022.1-MA-CF

11. SUPPLEMENTARY NOTES
The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This proceedings contains the 42 contributed papers which were selected from a total of 113 submitted extended abstracts. The program committee met in Arlington, Virginia, on January 21–22, 1994. The selection was based on the quality and originality of the results and their relevance to computational geometry. The papers generally represent preliminary reports of ongoing research and it is expected that most of them will eventually appear in more polished form in refereed journals.

14. SUBJECT TERMS

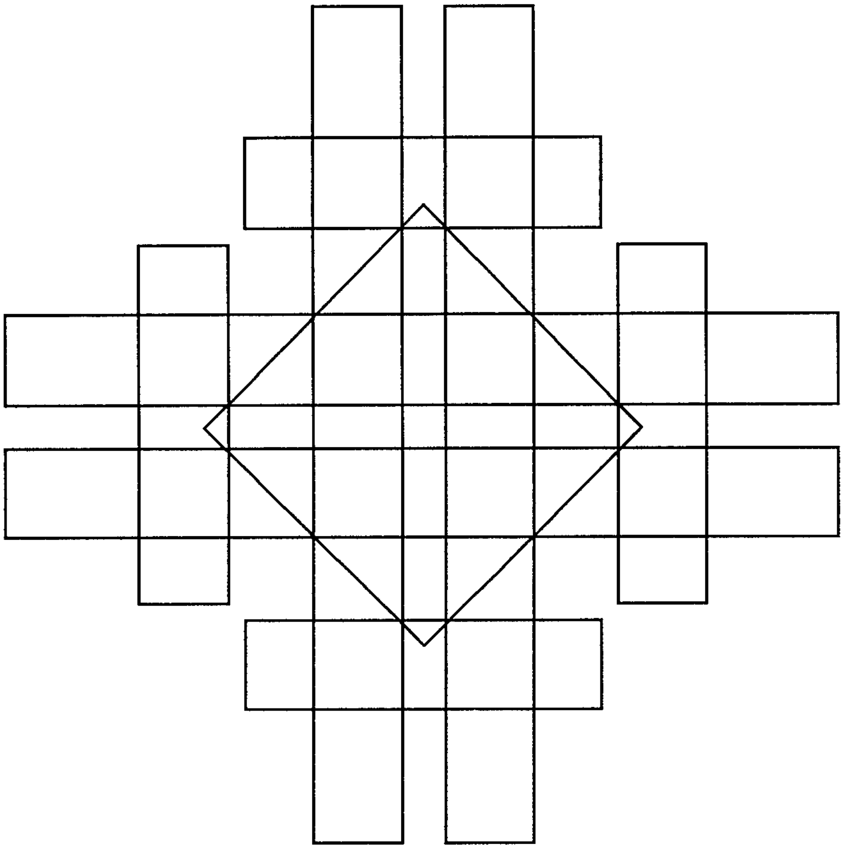
15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

PROCEEDINGS OF THE TENTH ANNUAL SYMPOSIUM ON

COMPUTATIONAL GEOMETRY



Stony Brook, New York
June 6-8, 1994

Sponsored by the ACM Special Interest Groups

for Graphics and Algorithms and Computation Theory

acm PRESS

The Association for Computing Machinery
1515 Broadway
New York, N.Y. 10036

ACM ISBN: 089791-648-4

Additional copies may be ordered prepaid from:

ACM Order Department
P.O. Box 12114
Church Street Station
New York, N.Y. 10257

Phone: 1-800-342-6626
(U.S.A. and Canada)
1-212-626-0500
(All other countries)
Fax: 1-212-944-1318
E-mail: acmpubs@acm.org

ACM Order Number: 429940

Printed in the U.S.A.

Foreword

This volume contains papers presented at the *Tenth Annual Symposium on Computational Geometry*, held June 6–8, 1994, in Stony Brook, New York. The Symposium was sponsored by the Special Interest Groups for Graphics (SIGGRAPH) and for Algorithms and Computation Theory (SIGACT) of the Association for Computing Machinery and was partially funded with the generous support of the agencies shown below.

This proceedings contains the 42 contributed papers which were selected from a total of 113 submitted extended abstracts. The program committee met in Arlington, Virginia, on January 21–22, 1994. The selection was based on the quality and originality of the results and their relevance to computational geometry. The papers generally represent preliminary reports of ongoing research and it is expected that most of them will eventually appear in more polished form in refereed journals. The program committee wishes to thank all people that helped us in refereeing the papers.

This year, for the second time, a video review of computational geometry was created. Descriptions of the accepted videos appear in these proceedings. For more details see the introduction on page 381.

The program committee wishes to thank all the authors who submitted abstracts in response to the call for papers.

Program Committee

Tetsuo Asano
Steve Fortune
Pat Hanrahan
Rolf Klein
Kurt Mehlhorn (Chair)
Ketan Mulmuley
Marco Pellegrini
Roberto Tamassia
Gert Vegter

Symposium Sponsored By

ACM Special Interest Groups for
Graphics (SIGGRAPH) and for
Algorithms & Computation Theory (SIGACT)

U.S. Army Research Office

Army Center for the Mathematics of
Nonlinear Systems at Stony Brook

Army Center of Excellence for Symbolic
Methods in Algorithmic Mathematics
(ACSyAM), Cornell University

Conference Chairs

Joseph S.B. Mitchell
Steven S. Skiena

Tenth Annual Symposium on Computational Geometry

June 6–8, 1994
State University of New York
Stony Brook, NY

Table of Contents

SESSION 1: CHAIRED BY TETSUO ASANO

SESSION 2: CHAIRED BY KETAN MULMULEY

SESSION 3: CHAIRED BY PAT HANRAHAN

SESSION 4: CHAIRED BY PAT HANRAHAN

SESSION 5: CHAIRED BY KURT MEHLHORN

SESSION 6: CHAIRED BY MARCO PELLEGRINI

SESSION 7: CHAIRED BY STEVEN FORTUNE

SESSION 8: CHAIRED BY STEVEN FORTUNE

SESSION 9: CHAIRED BY GERD VEGTER

SESSION 10: CHAIRED BY ROBERTO TAMASSIA

SESSION 11: CHAIRED BY ROLF KLEIN

VIDEO REVIEW

Vertical Decompositions for Triangles in 3-Space*

Mark de Berg[†] Leonidas J. Guibas[‡] Dan Halperin[§]

Abstract

We prove that, for any constant $\varepsilon > 0$, the complexity of the vertical decomposition of a set of n triangles in three-dimensional space is $O(n^{2+\varepsilon} + K)$, where K is the complexity of the arrangement of the triangles. For a single cell the complexity of the vertical decomposition is shown to be $O(n^{2+\varepsilon})$. These bounds are almost tight in the worst case.

We also give a deterministic output-sensitive algorithm for computing the vertical decomposition that runs in $O(n^2 \log n + V \log n)$ time, where V is the complexity of the decomposition. The algorithm is reasonably simple (in particular, it tries to perform as much of the computation in two-dimensional spaces as possible) and thus is a good candidate for efficient implementations.

1 Introduction

The study of arrangements plays a fundamental role in geometric computing. An arrangement is the partition of a Euclidean space into cells, as induced by a collection of possibly highly inter-penetrating objects. A surprising number of seemingly unrelated geometric problems boil down to the study of certain cells in such an arrangement. A famous example is the motion planning problem in robotics. Here the underlying arrangement is the arrangement in configuration space of the constraint surfaces defined by the obstacles and the robot. Because of these numerous applications, much research has been devoted to bounding the combinatorial complexity of arrangements, and of certain important subsets of arrangements such as zones and single cells.

For most algorithmic uses, however, a raw arrangement is an unwieldy structure. The difficulty is that cells in an arrangement can have very complex topologies, so navigating around them is difficult. What we often want is a further refinement of the cells into pieces, such as simplices, that are each homeomorphic to a ball and have constant description complexity. Ideally, the number of cells after the refinement should be proportional to the overall complexity of the arrangement. For arrangements of hyperplanes the well-known bottom vertex triangulation [12] meets this criterion. For more general arrangements such refined decompositions are more difficult to find. For example, for algebraic hypersurfaces of constant maximum degree in d-dimensional space ($d \geq 3$) the best decomposition technique known so far results in $O(n^{2d-3}\beta(n))$ cells [8], where $\beta(n)$ is an extremely slowly growing function,[1] whereas the complexity of the arrangement itself is only $O(n^d)$.

In this paper we study decompositions for arrangements of triangles in 3-dimensional space. The simplest way to decompose such an arrangement is to compute the bottom vertex triangulation of the arrangement of the planes containing the triangles. The resulting decomposition has size $\Theta(n^3)$, which is optimal in the worst case. In many applications, however, the actual complexity of the arrangement of triangles is much smaller. So the challenge is to obtain a decomposition whose size is sensitive to the complexity of the arrangement of the triangles.

Such a complexity-sensitive decomposition was given by Aronov and Sharir [2]: their *Slicing Theorem* states that one can decompose an arrangement of n triangles in 3-space into $O(n^2\alpha(n) + K)$ tetrahedra, where K is

*Mark de Berg was supported by the Dutch Organization for Scientific Research (N.W.O.), and by ESPRIT Basic Research Action No. 7141 (project ALCOM II:*Algorithms and Complexity*). Leonidas Guibas was supported by NSF grant CCR-9215219, by a grant from the Stanford SIMA Consortium, and by grants from the Digital Equipment, Mitsubishi, and Toshiba Corporations. Dan Halperin was supported by a Rothschild Postdoctoral Fellowship, by a grant from the Stanford Integrated Manufacturing Association (SIMA), by NSF/ARPA Research Grant IRI-9306544, and by NSF Grant CCR-9215219.

[†]Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands.

[‡]Department of Computer Science, Stanford University, Stanford, CA 94305.

[§]Robotics Laboratory, Department of Computer Science, Stanford University, Stanford, CA 94305.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

[1] To be precise: $\beta(n) = 2^{\alpha(n)^c}$, where c is a constant depending on the dimension and the degree of the surfaces. Here and throughout the paper, $\alpha(n)$ is the extremely slowly growing functional inverse of Ackermann's function.

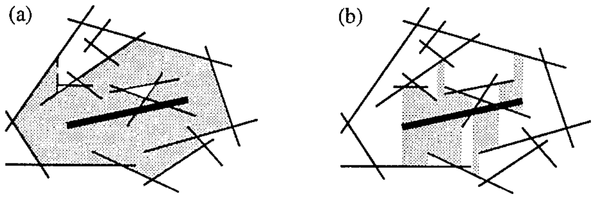


Figure 1: The vertical wall for the fat edge in the Slicing Theorem and in the vertical decomposition.

the complexity of the arrangement. This result is close to optimal: $\Omega(K)$ is clearly a lower bound on any decomposition, and Chazelle [5] shows that there are arrangements of complexity $O(n)$ such that any decomposition into convex cells has size $\Omega(n^2)$. (The triangles in Chazelle's example form the boundary of a simple polytope.) The Slicing Theorem obtains a decomposition by adding vertical walls for each of the triangle boundary edges, one after the other. The wall of an edge e is obtained by "flooding" the zone of e in an arrangement on the vertical plane $H(e)$ containing e; this arrangement is defined by intersections of $H(e)$ with the triangles and with already added walls. See Figure 1(a); the dashed segment in this figure is a previously added wall. After adding the walls one is left with convex cells that can easily be decomposed into tetrahedra. The Slicing Theorem decomposition has the unpleasant characteristic that it depends on the order in which triangle boundary edges are treated. Thus the tetrahedra in the decomposition are not defined "locally", and it is not canonical in the sense of Chazelle and Friedman [10]. This means that the decomposition cannot be used in randomized incremental algorithms. It also makes it difficult to compute the decomposition efficiently.

A decomposition which does not have this problem—and one which we think is more intuitive—is the following [11, 22, 23]. This decomposition is also obtained by erecting vertical walls. This time the wall for edge e simply consists of those points in $H(e)$ that can be connected to e with a vertical segment that does not cross any of the triangles in T. See Figure 1(b). This gives us a first decomposition $\mathcal{V}_1(T)$. Secondly, walls are erected from the intersection edges between pairs of triangles to produce a finer decomposition $\mathcal{V}_2(T)$. Observe that the wall erected from an edge is not obstructed by other walls, so the decomposition does not depend on the order in which the edges are treated. We call this decomposition the *vertical decomposition* for T and denote it by $\mathcal{V}(T) = \mathcal{V}_2(T)$. Note that the cells in $\mathcal{V}_2(T)$ need not be convex; in fact, they need not even be simply connected. But the decomposition can easily be refined into a convex subdivision $\mathcal{V}_3(T)$ where each cell has constant complexity, without increasing the asymptotic complexity of the subdivision—see below for details. We call the refined subdivision the *full vertical decomposition*[2] for T.

In this paper we prove bounds on the maximum combinatorial complexity of vertical decompositions, which are sensitive to the complexity of the arrangement of the triangles. More precisely, we show that, for any constant $\varepsilon > 0$, the complexity of the vertical decomposition of a set T of n triangles in 3-space is $O(n^{2+\varepsilon}+K)$, where K is the complexity of the arrangement $\mathcal{A}(T)$ induced by T. Our proof uses an interesting combination of efficient hierarchical cuttings [6, 21], the counting scheme used in hereditary segment trees [9], the Slicing Theorem [2], and random sampling [13, 19]. Our proof can be adapted to show that the vertical decomposition of a single cell in an arrangement of triangles has $O(n^{2+\varepsilon})$ complexity. We also give an example of a set T of triangles whose vertical decomposition has complexity $\Theta(n^2\alpha^2(n))$, whereas the complexity of $\mathcal{A}(T)$ is only $\Theta(n\alpha(n))$. This shows that our bound is close to optimal.

Secondly, we present a deterministic algorithm for constructing $\mathcal{V}(T)$ which runs in time $O(n^2 \log n + V \log n)$, where V is the complexity of $\mathcal{V}(T)$. The algorithm is reasonably simple (in particular, it tries to perform as much of the computation in two-dimensional spaces as possible) and thus is a good candidate for efficient implementations. The algorithm is also interesting as it is a three-dimensional version of a Bentley-Ottmann style sweep and may be adaptable to compute other partial or total information about the arrangement. Our approach is related to a space sweep algorithm that has recently been developed to compute a decomposition of certain arrangements for motion planning problems [17], and to the space sweep methods used to construct point location data structures for monotone subdivisions [14, 25].

2 Preliminaries

Let $T = \{t_1, \ldots, t_n\}$ be a collection of n (possibly intersecting) triangles in $\mathbb{R}^3$. Let $\mathcal{A}(T)$ denote the arrangement induced by T, namely, the subdivision of 3-space into cells of dimensions $0, 1, 2$ and 3, induced by the triangles in T. We assume that the triangles in T are in *general position.* (For details, see, e.g., [2]). By standard arguments (see, e.g., [26]) the combinatorial bound that we derive in Section 3 holds for degenerate arrangements as well. However, the algorithm described

[2] Mulmuley [23] calls $\mathcal{V}_3(T)$ the vertical decomposition, and he calls $\mathcal{V}_2(T)$ the cylindrical decomposition.

in Section 4 will have to undergo several technical adjustments (which we do not discuss in this paper) to accommodate for degenerate arrangements.

The *combinatorial complexity* (or *complexity* in short) of an arrangement $\mathcal{A}$ is defined to be the overall number of cells of various dimensions in $\mathcal{A}$; we denote the complexity of $\mathcal{A}$ by $|\mathcal{A}|$.

Central to the concept of vertical decompositions is the following notion of visibility: two points $p, q \in \mathbb{R}^3$ *(vertically) see each other with respect to* T if and only if the segment $\overline{pq}$ connecting them is vertical and the interior of $\overline{pq}$ does not intersect any triangle in T. Usually the set T is clear from the context and we just say that that p and q see each other. This definition is extended to objects other than points as follows: two sets $P, Q \subset \mathbb{R}^3$ see each other if and only if there are points $p \in P$, $q \in Q$ that see each other.

We define two three-dimensional entities related to a 3D curve γ. Let $H(\gamma)$ be the vertical surface that is the union of all the vertical lines which contain a point of γ. Let the *vertical wall extended from the 3D curve* γ, denoted $W(\gamma, T)$, be defined as follows: $W(\gamma, T) = \{p \in \mathbb{R}^3 : p \text{ sees } \gamma\}$. In other words, $W(\gamma, T)$ is the union of all vertical segments of maximal length that have a point of γ as an endpoint and whose interior does not intersect any triangle in T. Note that some of these segments can be rays.

3 The Combinatorial Bound

Let $T = \{t_1, \ldots, t_n\}$ be a set of triangles in $\mathbb{R}^3$, as defined above. We investigate the maximum combinatorial complexity of $\mathcal{V}(T) = \mathcal{V}_2(T)$ as a function of n, the number of triangles in T, and K, the complexity of $\mathcal{A}(T)$. Note that the complexity of $\mathcal{V}(T)$ is at least K.

We denote by $E(T)$ the set of segments in 3-space which are either an edge of a triangle in T or the intersection of two triangles in T. We call the segments in $E(T)$ edges; when we discuss edges of triangles in T we will explicitly say triangle boundary edges, and when we discuss intersections between triangles we will say intersection edges. For an edge $e \in E(T)$ we define its vertical wall to be $W(e, T)$. Let $\mathcal{W}(T) := \{W(e, T) : e \in E(T)\}$ be the collection of all the vertical walls. The vertical decomposition for T is the subdivision induced by the set $T \cup \mathcal{W}(T)$.

Let us first consider the complexity of a single wall $W(e, T)$. Recall that $H(e)$ is the vertical surface containing e. By definition, the part of $W(e, T)$ which is above e is bounded by the lower envelope of the segments which are intersections of the other triangles in T with $H(e)$ and which lie above e. See also Figure 1(b). It is well known that the complexity of the lower envelope of n segments in the plane is $O(n\alpha(n))$ [18]. A similar argument holds for the part of $W(e, T)$ below e. Hence, a single wall has complexity $O(n\alpha(n))$. Since there are $3n$ triangle boundary edges we have

Observation 3.1 *The total complexity of the walls $W(e, T)$ for all boundary edges e of the triangles in T is $O(n^2\alpha(n))$.*

Notice that the total number of edges in $E(T)$—and thus the total number of walls—is $O(n^2)$. It follows that the maximum complexity of $\mathcal{V}(T)$ is $O(n^3\alpha(n))$, as was noted in [23]. However, it is not clear whether it is possible that all walls have $\Theta(n\alpha(n))$ complexity. Indeed, below we show that this cannot happen when K is large. When $K = O(n\alpha(n))$, however, most walls can have large complexity.

Theorem 3.1 *There exists a set T of n triangles in $\mathbb{R}^3$ with $|\mathcal{A}(T)| = \Theta(n\alpha(n))$ and $|\mathcal{V}(T)| = \Theta(n^2\alpha^2(n))$.*

Proof: Let S' be a set of $\lfloor n/2 \rfloor$ line segments in the yz-plane whose upper envelope has complexity $\Theta(n\alpha(n))$ [28], and such that S' lies completely below the plane $z = 0$. Now extend each segment in the x-direction to obtain a set T' of infinitely long strips, that is, let $T' = \{[-\infty : \infty] \times s : s \in S'\}$. (The construction can easily be modified to use bounded triangles instead of infinitely long strips.) The upper envelope of T' consists of $\Theta(n\alpha(n))$ lines that are parallel to the x-axis. In the same way we can construct a set T'' of $\lceil n/2 \rceil$ strips whose *lower* envelope consists of $\Theta(n\alpha(n))$ lines that are parallel to the y-axis, and that lie completely above the plane $z = 0$. For the set $T = T' \cup T''$ we have $|\mathcal{A}(T)| = \Theta(n\alpha(n))$, and $|\mathcal{V}(T)| = \Theta(n^2\alpha^2(n))$. □

Next, we will establish an upper bound on the complexity of the vertical decompositions of sets of triangles in 3-space. We already know that the total complexity of the walls erected for the triangle boundary edges is $O(n^2\alpha(n))$ so now consider a wall erected from an intersection edge e. Let $S(e)$ be the set of segments which are the intersections of the other triangles with the vertical surface $H(e)$.

As remarked before, the part of $W(e, T)$ above e is bounded by the lower envelope of (the parts of) the segments in $S(e)$ lying above e, and the part of $W(e, T)$ below e is bounded by the upper envelope of (the parts of) the segments in $S(e)$ lying below e. The complexity of $W(e, T)$ is therefore linear in the number of points of the following types:[3]

1. endpoints of a segment $l \in S(e)$ which are contained in e

[3] If the wall has constant complexity this may not be true, but the total complexity of all constant complexity walls is $O(n^2)$.

2. endpoints of a segment $l \in S(e)$ which lie strictly above e and are vertically visible from e

3. intersections between two segments $l_1, l_2 \in S(e)$ which are vertically visible from e

The first type of feature is the intersection of the two triangles that define e and the triangle that defines l, that is, it is a vertex of $\mathcal{A}(T)$. Each vertex participates in a constant number of walls so the total number of type 1 features over all the walls is $O(K)$. For type 2 we note that the endpoint of l is the intersection of a boundary edge e' of the triangle that defines l with $H(e)$. So there is a visibility between e and e', which implies that e defines a feature of $W(e', T)$. Recall that the sum of the complexities of the walls erected from triangle boundary edges is $O(n^2\alpha(n))$. Hence, the total number of type 2 features is also $O(n^2\alpha(n))$. In the remainder of this section we bound the total number of type 3 features.

The bipartite case

We first study the following "bipartite" version of the problem. Let h be a fixed non-vertical plane, let T_1 be a set of n triangles lying completely below h, and let T_2 be a set of n triangles lying completely above h. We want to bound the number of pairs $e_1 \in E(T_1), e_2 \in E(T_2)$ that can see each other. Let $b(T_1, T_2)$ denote this number, and let $b(n)$ be the maximum value of $b(T_1, T_2)$ over all sets T_1 and T_2 of n triangles each, as defined above. Our example which proves Theorem 3.1 also shows that $b(n) = \Omega(n^2\alpha^2(n))$. We now establish upper bounds for $b(n)$. Recall that we only need to consider visibilities between intersection edges, as the remaining number of visibilities is $O(n^2\alpha(n))$.

We say that a planar curve γ is *convex* if and only if γ is contained in the boundary of its convex hull. In other words, any line intersects γ at most twice. The following simple lemma is crucial in our upper bound proof.

Lemma 3.1 *Let γ be a convex curve in the plane h. Then the number of visibilities between segments in $E(T_i)$ and γ is $O(n2^{\alpha(n)})$, for $i = 1, 2$.*

Proof: Recall that $H(\gamma)$ is the vertical surface containing γ. Define $t^* = t \cap H(\gamma)$ and $T_1^* = \{t^* : t \in T_1\}$. The curve γ sees a segment $e \in E(T_1)$ if and only if $e \cap H(\gamma)$ is a vertex of the upper envelope of T_1^*. The intersection $t_i \cap t_j$ of two triangles intersects $H(\gamma)$ at most twice. Hence, the upper envelope of the set $\{t \cap H(\gamma) : t \in T_1\}$ has complexity $\lambda_4(n) = O(n2^{\alpha(n)})$ [1]. A similar argument holds for $E(T_2)$. □

Using this lemma we can prove an almost tight upper bound on $b(n)$. A basic ingredient in the proof are efficient hierarchical cuttings [6, 21], which we define next. Let H be a set of n hyperplanes in $\mathbb{R}^d$. A $(1/r)$-cutting for H is a subdivision of $\mathbb{R}^d$ into disjoint simplices such that the interior of each simplex is intersected by at most n/r hyperplanes in H. The size of a cutting is the number of simplices it consists of. We say that a cutting Ξ' C-refines a cutting Ξ if every simplex of Ξ' is completely contained in a single simplex of Ξ and every simplex of Ξ contains at most C simplices of Ξ'. Now let C, ρ be constants and r a parameter with $1 \leq r \leq n$. A sequence $\Xi = \Xi_0, \Xi_1, \ldots, \Xi_k$ of cuttings is called an *efficient hierarchical* $(1/r)$*-cutting (for H)* if Ξ_0 is the single "simplex" $\mathbb{R}^d$, every Ξ_i $(1 \leq i \leq k)$ is a $(1/\rho^i)$-cutting of size $O(\rho^{id})$ which C-refines Ξ_{i-1}, and $\rho^{k-1} < r \leq \rho^k$. Notice that the last condition implies that $k = \Theta(\log r)$. We call the simplex in Ξ_{i-1} that contains a certain simplex $s \in \Xi_i$ the parent of s, denoted by parent(s).

Lemma 3.2 $b(n) = O(n^2 2^{\alpha(n)} \log n)$.

Proof: Project the triangles of T_1 and T_2 vertically onto the plane h. Construct an efficient hierarchical $(1/6n)$-cutting $\Xi = \Xi_0, \Xi_1, \ldots, \Xi_k$ for the $6n$ lines containing the projected triangle edges. For a simplex $s \in \Xi_i$, let $T_1^{\subset}(s) \subset T$ be the set of triangles in T_1 whose projection fully contains s but whose projection does not contain parent(s), and let $T_1^{\times}(s)$ be the set of triangles whose projection intersects s but does not contain it. Let $T_1(s) := T_1^{\subset}(s) \cup T_1^{\times}(s)$. Define $T_2^{\subset}(s)$, $T_2^{\times}(s)$, and $T_2(s)$ analogously. Finally, let $T^{\subset}(s) := T_1^{\subset}(s) \cup T_2^{\subset}(s)$, $T^{\times}(s) := T_1^{\times}(s) \cup T_2^{\times}(s)$ and $T(s) := T_1(s) \cup T_2(s)$. In the remainder we only work with "clipped" triangles, that is, for each triangle $t \in T(s)$ we only consider the part that projects onto s.

Consider a visibility between intersection edges $e_1 \in E(T_1)$ and $e_2 \in E(T_2)$. Let $t(e_1)$ and $t'(e_1)$ be the two triangles that define e_1, that is, $e_1 = t(e_1) \cap t'(e_1)$. Similarly, let $e_2 = t(e_2) \cap t'(e_2)$. We denote the projection of e_i onto h by $\overline{e_i}$. The basic observation behind our proof is the fact that there must be a simplex s in some cutting Ξ_i such that $\overline{e_1} \cap \overline{e_2} \in s$, $t(e_1), t'(e_1), t(e_2), t'(e_2) \in T(s)$, and at least one of these triangles is in $T^{\subset}(s)$. This implies that we only have to count for each simplex $s \in \Xi$ the number of visibilities where at least one of the involved triangles is in $T^{\subset}(s)$. (A similar observation is often made when one uses hereditary segment trees [9]: only long-long and long-short intersections need to be considered, not short-short intersections.)

So let us count the number of such visibilities for a given simplex s. Let $n_s := |T(s)|$. Consider some triangle $t \in T_1(s)$. We shall count the number of visibilities involving intersections of t and triangles $t' \in T_1^{\subset}(s)$. To this end we consider the intersection of t with the upper envelope of $T_1^{\subset}(s)$; an intersection between t and a part of some $t' \in T_1^{\subset}(s)$ that is not on the upper envelope can never be visible from above. The upper envelope of $T_1^{\subset}(s)$ is on the boundary of a convex polyhedron, and the projection of the intersection of t with the upper envelope of $T_1^{\subset}(s)$ is contained in a convex curve. Lemma 3.1 now tells us that the total number of visibilities involving intersections of t and some triangle $t' \in T_1^{\subset}(s)$ is $O(n_s 2^{\alpha(n_s)})$. A similar argument holds of course for the triangles in $T_2(s)$, so the total number of visibilities at s that involve at least one triangle in $T^{\subset}(s)$ is $O(n_s^2 2^{\alpha(n_s)})$.

To obtain the total number of visibilities we have to sum over all simplices s in the hierarchical cutting. Each triangle $t \in T(s)$ has an edge intersecting parent(s). Hence, for a simplex $s \in \Xi_i$ we have $n_s \leq n/\rho^{i-1}$. Thus the total number of visibilities can be bounded as follows:

$$\begin{aligned}
\textstyle\sum_s O(n_s^2 2^{\alpha(n_s)}) &= 2^{\alpha(n)} \textstyle\sum_{0\leq i\leq k} \sum_{s\in\Xi_i} O(n_s^2) \\
&= 2^{\alpha(n)} \textstyle\sum_{0\leq i\leq k} O(\rho^{2i}) \cdot O((n/\rho^{i-1})^2) \\
&= n^2 2^{\alpha(n)} \textstyle\sum_{0\leq i\leq k} O(1) \\
&= O(n^2 2^{\alpha(n)} \log n)
\end{aligned}$$

□

The general case

Before we can prove a bound on the complexity of vertical decompositions which is sensitive to the complexity of the arrangement, we need to prove the following worst-case bound.

Lemma 3.3 *The complexity of the vertical decomposition of a set of n triangles in $\mathbb{R}^3$ is $O(n^3)$.*

Proof: The proof is basically the same as the proof of Theorem 3.2 below, with the "intersection-sensitive" cuttings of Lemma 3.4 replaced by Chazelle's hierarchical cuttings. □

We need one more ingredient before we can prove the main theorem of this section.

Lemma 3.4 *Let T be a set of n triangles in $\mathbb{R}^3$ with $|\mathcal{A}(T)| = K$, and let r be a parameter with $1 \leq r \leq n$. There exists an $O(\log r/r)$-cutting of size $O(r^2\alpha(r) + Kr^3/n^3)$ for T.*

Proof: Let $R \subset T$ be a random sample of size r. By the Slicing Theorem [2] we can triangulate $\mathcal{A}(R)$ into $O(r^2\alpha(r) + |\mathcal{A}(R)|)$ simplices. By ε-net theory [19] each simplex will be intersected by $O(n \log r/r)$ triangles in T (with high probability). Finally, note that the expected complexity of $\mathcal{A}(R)$ is $O(r^2 + Kr^3/n^3)$. □

Theorem 3.2 *Let T be a set of n triangles in $\mathbb{R}^3$ with $|\mathcal{A}(T)| = K$. Then, for any constant $\varepsilon > 0$, the complexity of the vertical decomposition for T is $O(n^{2+\varepsilon} + K)$.*

Proof: Define $f(n,m)$ to be the maximum number of visibilities between edges in $E(T)$ over all sets T of n triangles where m is the number of triple intersections of the triangles in T. We shall prove that, for any $\varepsilon > 0$, $f(n,m) = O(n^{2+\varepsilon} + m)$.

Fix a large enough constant $r = r(\varepsilon)$. We distinguish two cases.

case (i): $m > n^3/r$

Because of Lemma 3.3 and r is a constant $f(n,m) = O(n^{2+\varepsilon} + m)$.

case (ii): $m \leq n^3/r$

Because $m \leq n^3/r$, Lemma 3.4 implies that there exists a $(c_1 \log r/r)$-cutting Ξ for T of size $c_2 r^2 \alpha(r)$, for some constants c_1, c_2. Let $T(s) \subset T$ denote the subset of triangles that intersect the interior of a simplex $s \in \Xi$. There are two types of visibilities between intersection edges of the triangles in T: visibilities such that there is a set $T(s)$ that contains all four triangles involved, and visibilities where there is no such set $T(s)$. The first type of visibilities is counted recursively. For visibilities of the second type we observe that the segment connecting the two intersection edges must intersect a facet of some simplex $s \in \Xi$. By Lemma 3.2 the number of such visibilities for a fixed facet is at most $c_3 n^2 2^{\alpha(n)} \log n$, for some constant c_3.

Define $n_s := |T(s)|$ and define m_s to be the number of triple intersections in $\mathcal{A}(T(s))$. Notice that $n_s \leq c_1 n \log r/r$ and $\sum_{s\in\Xi} m_s \leq m$, and that there are $4c_2 r^2 \alpha(r)$ facets of simplices in Ξ. We can now use induction to prove that there is a constant c such that $f(n,m) \leq n^{2+\varepsilon} + crm$:

$$\begin{aligned}
&f(n,m) \leq \\
&\leq 4c_2 r^2 \alpha(r) c_3 n^2 2^{\alpha(n)} \log n \;+\; \sum_{s\in\Xi} f(n_s, m_s) \\
&\leq 4c_2 r^2 \alpha(r) c_3 n^2 2^{\alpha(n)} \log n \;+\; \sum_{s\in\Xi} [n_s^{2+\varepsilon} + crm_s] \\
&\leq 4c_2 r^2 \alpha(r) c_3 n^2 2^{\alpha(n)} \log n \;+\; \sum_{s\in\Xi} (\tfrac{c_1 n \log r}{r})^{2+\varepsilon} \;+\; crm \\
&\leq n^{2+\varepsilon}[\tfrac{4c_2 r^2 \alpha(r) c_3 2^{\alpha(n)} \log n}{n^\varepsilon} + \tfrac{c_2 \alpha(r)(c_1 \log r)^\varepsilon}{r^\varepsilon}] \;+\; crm \\
&\leq n^{2+\varepsilon} + crm.
\end{aligned}$$

□

We apply a similar analysis to the case of a single cell. A single cell in an arrangement of n triangles can be decomposed into $O(n^2 \log n)$ simplices; this follows from the Slicing Theorem and a bound on the complexity of a single cell [3]. In the inductive analysis above we now have to recurse only on the $O(r^2 \log r)$ simplices of the single cell in the arrangement of a sample of size r, where in each subproblem we still deal with a single cell. We get the following result.

Corollary 3.1 *For any constant $\varepsilon > 0$, the maximum combinatorial complexity of the vertical decomposition of a single cell in an arrangement of n triangles in 3-space is $O(n^{2+\varepsilon})$.*

4 The Algorithm

Let $T = \{t_1, \ldots, t_n\}$ be a collection of triangles as defined in Section 2. To simplify the description of our algorithm, we will assume that the entire collection T of triangles is contained inside a *bounding simplex*, whose faces are special triangles in T. They are special in the sense that they violate the general position assumption. Also, unlike all the other triangles in T, we will be interested in only one side of each of these four triangles (the side that faces the interior of the simplex).

As mentioned in the introduction, we will consider a decomposition carried out in three steps: first we extend a vertical wall from every boundary edge of any triangle in T, thus we obtain $\mathcal{V}_1(T)$; in addition, we extend a vertical wall from every intersection edge of two triangles in T, and we get $\mathcal{V}_2(T)$; and finally we refine the subcells of $\mathcal{V}_2(T)$ into constant size subcells, to produce $\mathcal{V}_3(T)$.

The data structure that we obtain after carrying out the entire algorithm provides a comprehensive and convenient representation of the arrangement: each subcell in this representation has "constant description complexity"—it is bounded by up to six planar walls, which are quadrilaterals—it is homeomorphic to a ball, and the structure provides connectivity information between adjacent cells across vertical walls (we discuss the connectivity issue in detail in the full version of the paper). A significant advantage of the one-step decomposition $\mathcal{V}_1(T)$ is that in return for a relatively low overhead it already captures the 3D structure of the arrangement and it makes the next steps of the algorithm fairly simple.

For the most part, our algorithm solves two-dimensional subproblems and it mainly uses structures describing two-dimensional arrangements of segments. We will represent two-dimensional subdivisions using the quad-edge structure [16], and augment it with additional three-dimensional data as necessary.

Computing the decomposition $\mathcal{V}_1(T)$

As before, we denote the complexity of the arrangement $\mathcal{A}(T)$ by K. We start with computing the one-step decomposition of the arrangement. To transform $\mathcal{A}(T)$ into $\mathcal{V}_1(T)$ we have to add the vertical wall $W(e, T)$ for each boundary edge e of a triangle in T. In Observation 3.1 we have already seen that the complexity of all these walls is $O(n^2\alpha(n))$. Recall that the proof of this observation was based on the fact that each wall $W(e, T)$ is bounded from above (below) by the lower (upper) envelope of the intersections of other triangles with the vertical flat surface $H(e)$; see Figure 1(b). This fact also implies that we can compute a single wall in $O(n \log n)$ time [20], leading to the following lemma.

Lemma 4.1 *The set of walls $W(e, T)$ for all boundary edges e of the triangles in T can be computed in $O(n^2 \log n)$ time.*

For each triangle boundary edge e we split the portion between e and the upper boundary of $W(e, T)$ by exhibiting extra vertical edges emanating from each breakpoint of the upper boundary until they hit e (some of these edges are degenerate, since the corresponding breakpoint lies on e). We repeat the analogous procedure for the lower boundary, and represent the collection of vertices, edges and faces on $W(e, T)$ by a quad-edge structure. The structure size is proportional to the complexity of $W(e, T)$; it can be computed in time that is proportional to its size.

Next, we define a pair of two-dimensional arrangements of segments for each triangle t_i. We consider each triangle to be two-sided, and let t_i^- denote the side of t_i facing downward and t_i^+ be the side of t_i facing upward. We will use the notation t_i^* to refer to either side of t_i. For the bounding simplex, we need only one side of each triangle—the side facing the internal part of the simplex. The arrangement on t_i^+ is defined by a set $\Gamma(t_i^+)$ consisting of two types of segments: intersections of t_i with other triangles of T and boundary pieces of some $W(e, T)$ that lie on t_i^+. The second type of segments in $\Gamma(t_i^+)$ are, in other words, the contribution of t_i to the upper envelopes which bound some wall $W(e, T)$ from below. Similarly, the arrangement on t_i^- is defined by the set of segments $\Gamma(t_i^-)$ consisting of intersections of t_i with other triangles of T, and boundary pieces of some $W(e, T)$ that lie on t_i^-. For each segment in $\Gamma(t_i^*)$ we attach the label denoting its origin: either the label of the other triangle intersecting t_i or the edge e on whose envelope it appears. Let $n_i^* = |\Gamma(t_i^*)|$. We will use the abbreviation $\mathcal{A}_i^*$ to denote the two-dimensional arrangement $\mathcal{A}(\Gamma(t_i^*))$.

The next step of the algorithm is to compute, for each side of each triangle $t_i \in T$, the arrangement induced by $\Gamma(t_i^*)$. To this end we use a standard plane sweep algorithm that detects all the intersection points between segments[4] [24]. (At this point one may use the optimal algorithm by Chazelle and Edelsbrunner [7]. However, due to other steps of the algorithm, this will not make a difference in the overall running time of the algorithm. Also, the sweep paradigm is more appropriate for the generalization that we propose in the sequel.)

Lemma 4.2 *After having computed all walls of triangle boundary edges, all the two-dimensional arrangements $\mathcal{A}_i^*$ can be computed in time $O(|\mathcal{V}_1(T)|\log n)$.*

Again, we represent each two-dimensional arrangement by a quad-edge structure.

It is easy to verify that after the first step of the decomposition, the entire three-dimensional arrangement is connected, that is, there are no "floating" parts. In particular, the two-dimensional boundary of each three-dimensional cell is connected. The shape of each 3D cell may still be rather convoluted; it need not even be xy-monotone. Therefore, instead of handling each 3D cell at a time we carry out a space sweep over the entire decomposition $\mathcal{V}_1(T)$. (A similar approach has recently been used to obtain a decomposition of certain arrangements related to a motion planning problem [17]; see also [14, 25] for dynamic maintenance of a monotone subdivision in a space-sweep.)

Let P_{y_1} denote the plane $y = y_1$. Let $\mathcal{A}_{y_1} = \mathcal{A}(P_{y_1} \cap \mathcal{V}_1(T))$ be the two-dimensional arrangement of segments induced on the plane P_{y_1} by intersecting it with[5] $\mathcal{V}_1(T)$. We use $\mathcal{A}_y$ to denote this arrangement for an arbitrary y-value. We will be using the following property of the one-step decomposition (whose simple proof is omitted here).

Lemma 4.3 *Every face in $\mathcal{A}_y$ is x-monotone, that is, every z-vertical line intersects any face of $\mathcal{A}_y$ in at most one connected component.*

Transforming $\mathcal{V}_1(T)$ into $\mathcal{V}_2(T)$

We now proceed to describe the second step of the algorithm. The major difficulty in transforming $\mathcal{V}_1(T)$ into $\mathcal{V}_2(T)$ is to efficiently detect the vertical visibilities of pairs of intersection edges, one on the ceiling of a cell of $\mathcal{V}_1(T)$ and the other on the floor of that cell. To this end we perform a space sweep with a plane P_y parallel to the xz-plane over $\mathcal{V}_1(T)$, from $y = -\infty$ to $y = +\infty$. Throughout the sweep we maintain dynamic data structures that describe $\mathcal{A}_y$. (Below we will indicate which data structures we need.) Roughly, our goal is to update each face of any arrangement $\mathcal{A}_i^+$ with all the edges of ceiling faces that are visible when looking vertically upward from that face, and similarly, to update each face of any arrangement $\mathcal{A}_i^-$ with all the edges of floor faces that are visible when looking vertically downward from that face.

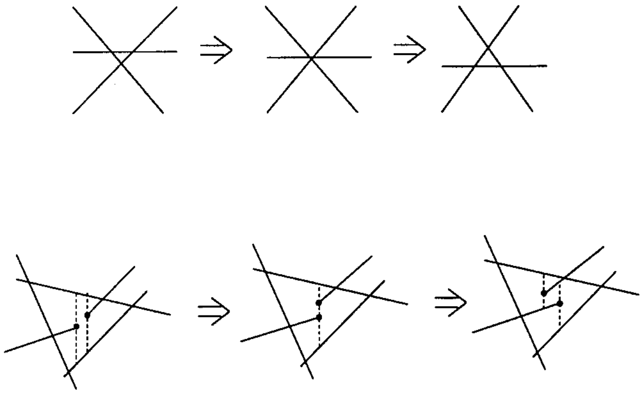


Figure 2: Examples of changes to $\mathcal{A}_y$ when sweeping over a vertex of $\mathcal{V}_1(T)$.

The arrangement $\mathcal{A}_y$ changes continuously as we sweep the plane P_y. At a finite number of "events", however, the combinatorial structure of $\mathcal{A}_y$ changes. It is not difficult to see that these events are exactly the vertices of $\mathcal{V}_1(T)$. At such an event the following changes to $\mathcal{A}_y$ can occur: (i) a vertex of $\mathcal{A}_y$ may (dis)appear; (ii) an edge of $\mathcal{A}_y$ may (dis)appear; and (iii) a face of $\mathcal{A}_y$ may (dis)appear. In fact, several such changes occur simultaneously at each event. Two examples are given in Figure 2. By the general position assumption, each event involves only a constant number of changes to a constant number of faces in $\mathcal{A}_y$.

Since our goal is to detect vertical visibilities between two (intersection) edges in $\mathcal{V}_1(T)$, we wish to detect when there is a vertical segment connecting two intersection edges that does not intersect any triangle in T. Consider the sweep plane P_y when it contains such a vertical segment connecting intersection edges e_1 and e_2. Then the intersection of e_1 and e_2 with P_y must define two vertices of $\mathcal{A}_y$ which are on the boundary of the same face and lie on a common vertical line. Thus we define a new type of event in our space sweep, called a *vertical* event, which occurs when two vertices of the

[4] From this point on, when discussing two-dimensional arrangements, we will distinguish between a segment and an edge. A *segment* is a maximal portion of a line that appears in the arrangement, possibly intersected by other segments. An *edge* is a maximal portion of a segment not intersected by any other segment.

[5] With a slight abuse of notation, we use $\mathcal{V}_1(T)$ to denote the subdivision of 3-space after the one-step decomposition, and also to denote the collection of triangles and walls that induce this subdivision.

same face become aligned along a vertical line.

We maintain all the events in a priority queue $\mathcal{Q}$, ordered by increasing y-coordinate. The operations we will perform on $\mathcal{Q}$ are *insert* an event, *delete* an event, and *fetch* the next event, that is, fetch the event with minimum y-value (we delete each event after it has been handled). We also need to perform membership check of an event in the queue, to avoid inserting the same event several times. Such a queue can be implemented so that the time for each operation is $O(\log m)$, where m is the maximum number of events held simultaneously in the queue [24]. Observe that we can insert all the events where the structure of $\mathcal{A}_y$ changes—the vertices of $\mathcal{V}_1(T)$—into $\mathcal{Q}$ before we start the sweep. To each event that we insert into the queue, we attach some local geometric and combinatorial information relevant to that event. The vertical events, however, have to be computed on the fly. Next we describe the data structures which we need to represent $\mathcal{A}_y$ in order to be able to compute the vertical events.

Let $f = f(y)$ be a face of $\mathcal{A}_y$. Recall that f is x-monotone. We split the vertices on the boundary of f into two x-monotone chains: the lower chain $L(f)$, and the upper chain $U(f)$. The vertices in each chain are ordered by increasing x-coordinate, and we implement each chain as a balanced binary tree. Whenever a vertex on the boundary of f disappears or newly appears, we update the relevant chain by deletion or insertion respectively. When an existing face disappears, we "free" its attached chains. When a face newly appears, we allocate a pair of new structures for its lower and upper chains. When a face f is split into two faces (for instance when an endpoint of a segment penetrates the face) we create new pairs of chains for these faces. These are easily constructed by splitting the chains of f and adding a small number of new vertices. Similarly, when two faces merge into one, we create the two chains of the new face by *join* operations on the chains of the merged faces plus possibly a few deletions of vertices. All these operations (insertion, deletion, split and join) can be carried out in $O(\log n)$ time each, using red-black trees [15], [27, Chapter 4].

To detect vertical visibilities between vertices on the boundary of f we proceed as follows. Whenever a new vertex v is created on the lower chain of f (the operations for a new vertex on the upper chain are symmetric), we look for its "neighbors" in the upper chain of f, that is, we look for the two vertices of $U(f)$ whose projection onto the x-axis lie nearest to the projection of v onto the x-axis. Let u_1 and u_2 be the two neighbors of v on the upper chain. Each vertex w on the boundary of f is the cross-section of P_y with an edge $e(w)$ of some $\mathcal{A}_i^*$. (It is important here that $e(w)$ is an edge of $\mathcal{A}_i^*$, not a segment on t_i^*. See footnote 4 for the difference.) Since we have attached to each vertex some additional information, we can compare $e(v)$ with each of $e(u_1)$ and $e(u_2)$ to see if their projections onto the xy-plane intersect. If they intersect, we have detected a potential vertical visibility between two intersection edges; we add this event to $\mathcal{Q}$, with the y-coordinate of the intersection point. We say *potential* because when we will come to handle this event, we may find that this "vertical visibility" is obscured by another triangle (or triangles) in a manner that we were unable to predict when the event was inserted into the queue. Therefore, we distinguish between two types of events: *actual* events, which are either the events corresponding to features of $\mathcal{V}_1(T)$, or additional events that indeed correspond to a vertical visibility, and *false* events, which are potential vertical visibilities that are found out to be obscured when handled.

When we handle an actual event q which corresponds to vertical visibility of a pair of vertices u and v on the lower and upper chains respectively, each of u and v now have new neighbors on the reciprocal chain, so we check these new neighbors for potential additional events as above.

The next lemma proves the correctness of our approach.

Lemma 4.4 *The set of events that is computed during the sweep contains the set of visibilities between pairs of intersection edges. The false events can be efficiently distinguished from the actual events. Each false event can be charged to an actual event, and no actual event gets charged more than a constant number of times this way.*

Proof: First we claim that no vertical visibility is missed by our algorithm. The reason is that before a pair of edges of $\mathcal{V}_1(T)$ become vertically visible, their corresponding cross-sections must become neighbors (in the sense defined above) in $L(f)$ and $U(f)$ for some face f of $\mathcal{A}_y$. This is similar to a basic argument in the Bentley-Ottmann algorithm for detecting intersections of line segments [4], [24, Section 7.2].

Once we handle a vertical visibility event it is easy to determine whether it is actual or false by checking whether the lower and upper chains involved in this event belong to the same face. The event is actual if and only if both chains belong to the same face. This could be done, for example, by maintaining cross pointers between the roots of the trees describing the chains.

We charge each false event q' to the actual event q that has "spawned" it. Every vertical visibility event is added to the queue $\mathcal{Q}$ only at actual events (a false event does not create new events). Since no event creates more than a constant number of additional events, no actual event will be charged more than a constant number of times for false events.

□

To represent the decomposition $\mathcal{V}_2(T)$, we augment the two-dimensional arrangements $\mathcal{A}_i^*$ with information on all the edges that are vertically visible from $\mathcal{A}_i^*$. More precisely, we add to $\mathcal{A}_i^+$ all the edges that are visible when looking vertically upward from t_i^+. Similarly, we add to $\mathcal{A}_i^-$ all the edges that are visible when looking vertically downward from t_i^-. We also record on these arrangements the intersection of these edges with the original edges of the corresponding $\mathcal{A}_i^*$. Note that these intersection points correspond exactly to the events computed during the space sweep.

Lemma 4.5 *The space sweep algorithm described above for transforming $\mathcal{V}_1(T)$ into $\mathcal{V}_2(T)$ runs in time $O(V \log n)$, where $V = |\mathcal{V}_2(T)|$.*

Proof: We have seen that every event can be handled with a constant number of operations on some chains $U(f)$ and $L(f)$, and on $\mathcal{Q}$. Thus every event takes $O(\log n)$ time. It remains to observe that every event is either a feature of $\mathcal{V}_2(T)$, or it can be charged to a feature of $\mathcal{V}_2(T)$ such that no feature of $\mathcal{V}_2(T)$ gets charged more than a constant number of times (Lemma 4.4). □

We denote the arrangement $\mathcal{A}_i^*$ after updating it in the above process by $\mathcal{B}_i^*$. Every face on every $\mathcal{B}_i^*$ represents a vertical cylindrical cell which has the same xy-projection as that face, and has a unique triangle bounding it on the top and a unique triangle bounding it on the bottom.

Transforming $\mathcal{V}_2(T)$ into $\mathcal{V}_3(T)$

The final step of our algorithm is to refine each $\mathcal{B}_i^*$ further to obtain the full vertical decomposition, in the following standard manner. Consider the projection of $\mathcal{B}_i^*$ onto the xy-plane. We extend a y-vertical segment from each vertex of the projected arrangement upward and downward until it reaches another segment, or the boundary of the projection of t_i. By projecting the segments we have added back to t_i^*, we obtain a refinement of $\mathcal{B}_i^*$ into trapezoids. Finally, we extend each of these newly added segments into vertical walls inside the respective three-dimensional cells to obtain $\mathcal{V}_3(T)$. Adding the y-vertical extensions to all the arrangements $\mathcal{B}_i^*$ can be carried out by a plane sweep of each of the arrangements $\mathcal{B}_i^*$. This last step of the decomposition, where we transform $\mathcal{V}_2(T)$ into $\mathcal{V}_3(T)$, does not increase the asymptotic complexity of the decomposition or the algorithm.

For lack of space, several details and proofs have been omitted throughout—they are discussed in the full version of the paper. The main algorithmic result of the paper is summarized in the following theorem:

Theorem 4.1 *Given a collection T of n triangles in general position in three-dimensional space, the time needed to compute the full vertical decomposition of the arrangement $\mathcal{A}(T)$ is $O(n^2 \log n + V \log n)$, where V is the combinatorial complexity of the vertical decomposition.*

Remark It is, in fact, possible to compute vertical decompositions with a somewhat lower overhead term than the algorithm above achieves, using rather involved multi-level data structures. The alternative approach is described in the full version of the paper.

5 Concluding Remarks

We have proved bounds on the maximum combinatorial complexity of the vertical decomposition of sets of triangles in 3-space that are sensitive to the complexity of the arrangement of triangles. We have also given a simple deterministic output-sensitive algorithm for computing the vertical decomposition.

While there are various technical details that still need to be verified, we believe that our algorithm will work with roughly the same time bound for the case of low-degree algebraic surface patches in 3-space. We are currently studying the full details of this extension. The generalization of the combinatorial result seems more difficult and currently we do not see how to adapt our proof to this case.

Another open problem is to tighten the small gap that remains between the upper and lower bounds for vertical decompositions. We believe that the n^ε-factor in the upper bound is only an artifact of our proof technique and that the lower bound is closer to the truth than the upper bound.

References

[1] P. K. Agarwal, M. Sharir, and P. Shor. Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences. *J. Combin. Theory Ser. A*, 52:228–274, 1989.

[2] B. Aronov and M. Sharir. Triangles in space or building (and analyzing) castles in the air. *Combinatorica*, 10(2):137–173, 1990.

[3] B. Aronov and M. Sharir. Castles in the air revisited. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 146–156, 1992.

[4] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.*, C-28:643–647, 1979.

[5] B. Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM J. Comput.*, 13:488–507, 1984.

[6] B. Chazelle. An optimal convex hull algorithm and new results on cuttings. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 29–38, 1991.

[7] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39:1–54, 1992.

[8] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. A singly-exponential stratification scheme for real semi-algebraic varieties and its applications. In *Proc. 16th Internat. Colloq. Automata Lang. Program.*, volume 372 of *Lecture Notes in Computer Science*, pages 179–192. Springer-Verlag, 1989.

[9] B. Chazelle, H. Edelsbrunner, L. J. Guibas, and M. Sharir. Lines in space: combinatorics, algorithms, and applications. In *Proc. 21st Annu. ACM Sympos. Theory Comput.*, pages 382–393, 1989.

[10] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[11] K. Clarkson, H. Edelsbrunner, L. Guibas, M. Sharir, and E. Welzl. Combinatorial complexity bounds for arrangements of curves and spheres. *Discrete Comput. Geom.*, 5:99–160, 1990.

[12] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17:830–847, 1988.

[13] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[14] M. Goodrich and R. Tamassia. Dynamic trees and dynamic point location. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 523–533, 1991.

[15] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, Lecture Notes in Computer Science, pages 8–21, 1978.

[16] L. J. Guibas and J. Stolfi. Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams. *ACM Trans. Graph.*, 4:74–123, 1985.

[17] D. Halperin and M. Sharir. Near-quadratic bounds for the motion planning problem for a polygon in a polygonal environment. In *Proc. 34th Annu. Sympos. on Foundations of Computer Science*, pages 382–391, 1993.

[18] S. Hart and M. Sharir. Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes. In *Proc. 25th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 313–319, 1984.

[19] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[20] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Inform. Process. Lett.*, 33:169–174, 1989.

[21] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 276–285, 1992.

[22] K. Mulmuley. Hidden surface removal with respect to a moving point. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 512–522, 1991.

[23] K. Mulmuley. Randomized multidimensional search trees: further results in dynamic sampling. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 216–227, 1991.

[24] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York, NY, 1985.

[25] F. P. Preparata and R. Tamassia. Efficient point location in a convex spatial cell-complex. *SIAM J. Comput.*, 21:267–280, 1992.

[26] M. Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. In *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS 93)*, pages 498–507, 1993.

[27] R. E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[28] A. Wiernik and M. Sharir. Planar realizations of nonlinear Davenport-Schinzel sequences by segments. *Discrete Comput. Geom.*, 3:15–47, 1988.

Almost Tight Upper Bounds for the Single Cell and Zone Problems in Three Dimensions*

Dan Halperin[†] Micha Sharir[‡]

Abstract

We consider the problem of bounding the combinatorial complexity of a single cell in an arrangement of n low-degree algebraic surface patches in 3-space. We show that this complexity is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on ε and on the maximum degree of the given surfaces and of their boundaries. This extends several previous results, almost settles a 7-year-old open problem, and has applications to motion planning of general robot systems with three degrees of freedom. As a corollary of the above result, we show that the overall complexity of all the three-dimensional cells of an arrangement of n low-degree algebraic surface patches, intersected by an additional low-degree algebraic surface patch σ (the so-called *zone* of σ in the arrangement) is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on ε and on the maximum degree of the given surfaces and of their boundaries.

1 Introduction

Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ be a given collection of n low-degree algebraic surface patches in 3-space (see below for a more precise statement of the properties that these surfaces are assumed to satisfy). We denote by $\mathcal{A}(\Sigma)$ the *arrangement* of Σ, i.e., the decomposition of 3-space into (relatively open) cells of various dimensions, each being a maximal connected set contained in the intersection of a fixed subcollection of Σ and not meeting any other surface; we will denote j-dimensional cells of $\mathcal{A}(\Sigma)$, for $j = 0, 1, 2$, as *vertices, edges, and faces*, respectively, and the unquantified term *cell* will be used to denote 3-dimensional cells of $\mathcal{A}(\Sigma)$. The *combinatorial complexity* of a cell C is the number of lower-dimensional cells appearing on its boundary. The problem studied in this paper is to obtain a sharp upper bound on the combinatorial complexity of a single cell in such an arrangement.

One of the main motivations for studying this problem is its applications to *robot motion planning.* Let B be an arbitrary robot system with 3 degrees of freedom, moving in some environment V filled with obstacles. Any placement of B can be represented by a point in 3-space, whose coordinates are the 3 parameters controlling the degrees of freedom of B; this space is called the *configuration space* of B. We want to compute the *free* portion of this space, denoted as *FP*, and consisting of those placements of B at which it does not meet any obstacle. We note that the boundary of *FP* consists of placements at which B makes contact with some obstacles, but does not penetrate into any of them. Under reasonable assumptions concerning B and V, we can represent the subset of 'contact placements' of B (including those placements at which B makes contact with an obstacle but may also penetrate into other obstacles) as the union of a collection of a finite number of surface patches, all algebraic of constant maximum degree (and whose relative boundaries are also algebraic of constant maximum degree).

For example, if B is an arbitrary polygonal object with k sides, and V is an open planar polygonal region bounded by m edges, the configuration space of B is a 3-dimensional space, each point of which represents a possible placement of B by the parametrization $(x, y, \tan\frac{\theta}{2})$, where (x, y) are the coordinates of some fixed reference point on B, and θ is the orientation of B. In this case, each 'contact surface' is either the locus of all placements of B at which some specific corner of B touches some specific edge of V, or the locus of

*Work on this paper by the first author has been supported by a Rothschild Postdoctoral Fellowship, by a grant from the Stanford Integrated Manufacturing Association (SIMA), by NSF/ARPA Research Grant IRI-9306544, and by NSF Grant CCR-9215219. Work on this paper by the second author has been supported by NSF Grant CCR-91-22103, and by grants from the U.S.-Israeli Binational Science Foundation, the G.I.F., the German-Israeli Foundation for Scientific Research and Development, and the Israel Science Fund administered by the Israeli Academy of Sciences.

[†] Robotics Laboratory, Department of Computer Science, Stanford University.

[‡] School of Mathematical Sciences, Tel Aviv University, and Courant Institute of Mathematical Sciences, New York University.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

placements at which some side of B touches some vertex of V. Each of the resulting $O(km)$ contact surfaces is a 2-dimensional algebraic surface patch of degree at most 4, and its relative boundary consists of a constant number of algebraic arcs, of constant maximum degree as well.

If B is placed at a free placement Z and moves continuously from Z, then it remains free as long as the corresponding path traced in configuration space does not hit any contact surface. Moreover, once this path crosses a contact surface, B becomes non-free (assuming, as is customary, *general position* of B and V). It follows that the connected component of *FP* that contains Z is the cell that contains Z in the arrangement $\mathcal{A}(\Sigma)$ of the contact surfaces. (The entire *FP* is the union of a collection of certain cells in this arrangement.) Hence, bounding the combinatorial complexity of a single cell in such an arrangement is a major problem one has to tackle, prior to the design of efficient algorithms for computing such a cell.

Here is a brief history of the single-cell problem. In two dimensions, it has been shown in [14] that the complexity of a single face in an arrangement of n Jordan arcs, each pair of which intersect in at most some constant number, s, of points, is $O(\lambda_{s+2}(n))$, where $\lambda_q(m)$ is the maximum length of *Davenport-Schinzel sequences* of order q composed of m symbols, and is nearly linear in m for any fixed q (see [1, 18] for more details). Thus the maximum complexity of a single face is nearly linear in the number of arcs (for any fixed s), as opposed to the complexity of the entire arrangement of the arcs, which can be quadratic in the worst case. Efficient algorithms for computing a single face in a 2-D arrangement are given in [14] and in [8].

In higher dimensions, a prevailing conjecture (see, e.g., [22]) is that the complexity of a single cell in an arrangement $\mathcal{A}(\Sigma)$ as above is at most close to $O(n^{d-1})$, which is again roughly one order of magnitude smaller than the maximum complexity of the entire arrangement, which can be $\Theta(n^d)$ (see [23]). A stronger version of the conjecture asserts that the maximum complexity of a single cell in such an arrangement is $O(n^{d-2}\lambda_s(n))$, where s is some constant that depends on the maximum degree of the given surfaces and of their boundaries.

These conjectures have been proved only for a few special cases of arrangements. They are largely open in the general case stated above. In fact, no bounds better than $O(n^d)$ are known for the general case, even in three dimensions. The special cases for which better bounds are known include the case of hyperplanes, where the complexity of a single cell, being a convex polytope bounded by at most n hyperplanes, is $O(n^{\lfloor d/2 \rfloor})$ (by the Upper Bound Theorem [21]), the case of spheres, where an $O(n^{\lceil d/2 \rceil})$ bound is easy to obtain by lifting the spheres into hyperplanes in $(d+1)$-space [11, 24], the case of $(d-1)$-simplices, where an $O(n^{d-1}\log n)$ bound has been established in [5], and several special cases in three dimensions that arise in motion planning for various specific robot systems B with three degrees of freedom, including the case of a moving polygon mentioned above, where an $O(n^{2+\varepsilon})$ bound is proved in [17], some restricted cases of that problem, where slightly more improved bounds are obtained [15], and a few other systems (see [15]).

The single cell problem is a generalization of the related problem of bounding the complexity of the *lower envelope* of Σ, i.e., the portion of the union of the surfaces of Σ, consisting of those points w for which no surface of Σ passes below w. This problem, also rather difficult, is nevertheless easier to analyze, and recent results [16, 26] show that the combinatorial complexity of such an envelope is $O(n^{d-1+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on ε, d, and the maximum algebraic degree of the given surfaces and of their relative boundaries.

In this paper we derive an improved upper bound for the complexity of a single cell in an arrangement $\mathcal{A}(\Sigma)$ of algebraic surfaces in 3-space, as above. This bound, in 3 dimensions, is the same as the bound for lower envelopes just mentioned; that is, it is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends, as above, on ε and on the maximum algebraic degree of the given surfaces and of their relative boundaries. This almost establishes the conjecture mentioned above in three dimensions.

Our analysis adapts the proof technique of [17], which in turn is based on the analysis technique of [16, 26] for the case of lower envelopes. The lesson one can learn from the analysis in [17] is that in the case of a single cell one needs the following two preliminary results to 'bootstrap' the recurrences appearing in the analysis:

(a) a sharp bound on the number of 'x-extreme' vertices of the cell C (vertices whose x coordinate is smallest or largest in a small neighborhood of the vertex within the closure of C), and

(b) a sharp bound on the number of vertices bounding 'popular' faces of C (faces that are adjacent to C on both 'sides'; see [3, 5, 17] and below).

Bounds on these quantities were obtained in [17] using special properties of the surfaces that arise in the case studied there. A main technical contribution of the present paper is a derivation of such bounds in the general algebraic setting assumed above. The bound (a) is obtained using considerations which are related to Morse theory (see e.g. [20]), but are simpler to derive in 3 dimensions. The bound (b) is obtained by applying the new probabilistic technique of [16, 17, 26] to counting only the vertices of popular faces (this idea is in the spirit of the methodology used in [3, 5]). Once these two bounds are available, the rest of the proof is rather similar to those used in [16, 17, 26], although certain non-trivial adjustments are required.

The paper is organized as follows. In Section 2 we give several preliminary results, including the analysis of the number of x-extreme vertices of a single cell. The main analysis is presented in Section 3. The application of the main result to the *zone* problem, as mentioned in the abstract, is discussed in Section 4, and the paper concludes in Section 5 with further applications of our results and some open problems.

2 Preliminaries

Let $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ be a given collection of n surface patches in 3-space that satisfy the following conditions:

(i) Each σ_i is monotone in the xy-direction (that is, every vertical line intersects σ_i in at most one point). Moreover, each σ_i is a portion of an algebraic surface of constant maximum degree b.

(ii) The vertical projection of σ_i onto the xy-plane is a planar region bounded by a constant number of algebraic arcs of constant maximum degree (say, b too).

(iii) The relative interiors of any triple of the given surfaces intersect in at most s points (by Bezout's theorem [19] and by Property (iv) below, we always have $s \leq b^3$).

(iv) The surface patches in Σ are in *general position*; one way of defining this is to require that the coefficients of the polynomials defining the surfaces and their boundaries are algebraically independent over the rationals (i.e., no multivariate polynomial with rational coefficients vanishes when substituting into it some of the given coefficients), thereby excluding all kinds of 'degenerate' configurations; see [16, 17, 26] for more details.

We remark that the somewhat restrictive condition (iv) and the first part of condition (i) are not essential for the analysis. If the first part of condition (i) does not hold, we can decompose each surface into a constant number of xy-monotone pieces by cutting it along the locus of points with z-vertical tangency. If condition (iv) does not hold, we can argue, by applying, as in [26], a small random perturbation of the polynomials, that the complexity of a single cell in a degenerate arrangement of surfaces is at most proportional to the worst-case complexity of a single cell in arrangements of surfaces in general position.

We are also given a point Z not lying on any surface, and define $C = C_Z(\Sigma)$ to be the cell of the arrangement $\mathcal{A}(\Sigma)$ that contains Z; by definition, C is an open set in $\mathbb{R}^3$.

Theorem 2.1 *The number of x-extreme vertices of C is $O(n^2)$.*

Proof. For each $x_0 \in \mathbb{R}$, let π_{x_0} denote the plane $x = x_0$. We call a point $w \in \partial C$ *critical* if there exists a neighborhood N of w and a connected component K of $C \cap N$ so that $K \cap \pi_{x_0}$ is disconnected, where x_0 is the x-coordinate of w, but $K \cap \pi_x$ is connected either for all $x < x_0$ or for all $x > x_0$ sufficiently close to x_0.

We first claim that the number of x-extreme vertices of C is proportional to 2 plus the number of critical points of C. We prove this by an argument borrowed from [6].

We sweep C by moving the plane π_x in the direction of increasing x, and keep track of the number I of connected components of $C \cap \pi_x$. This number is initially (at $x = -\infty$) $O(n^2)$, since this is an upper bound on the overall complexity of any planar cross section of the entire arrangement $\mathcal{A}(\Sigma)$. I increases by 1 when π_x sweeps through a local minimum of C, or when a connected component of $C \cap \pi_x$ splits into two subcomponents; I decreases by 1 when π sweeps through a local maximum of C, or when two components of $C \cap \pi_x$ merge into a single component. (The general position assumption implies that only two components can merge into, or split from, a component of C at any given x.) The number of events at which components can split or merge is equal, by definition, to the number Q of critical points of C.

Consider the following dynamic scheme for assigning weights to each component of $C \cap \pi_x$. Initially, at $x = -\infty$, we assign weight -1 to each component of $C \cap \pi_x$. When π_x sweeps through a local minimum point of C, a new component of $C \cap \pi_x$ is created, and is also assigned weight -1. When two components of $C \cap \pi_x$ merge, we assign to the new component weight 2 plus the sum of the weights of the merged components. When a component shrinks and disappears, its final weight is added to a global count M. When a component is split into two subcomponents, each of them is assigned weight $1 + \frac{w}{2}$, where w is the weight of the split component.

We claim that, at any given time during the sweep, the weight of any component of $C \cap \pi_x$ is always at least -1, and the weight of a component that was formed by one or more preceding splitting and merging operations is non-negative. Both claims are easy to prove by induction on the sweep events. If C has no critical points then it has at most one local minimum and at most one local maximum, and the claim holds trivially in this case; so suppose C does have critical points. In this case it is easily verified, using induction on the sweep events, that the weight of each component of $C \cap \pi_x$ that shrinks to a point as we reach a local maximum of C is nonnegative, so the value of M is always nonnegative. Similarly, all components that survive as x reaches $+\infty$ have nonnegative weight.

Suppose that, at some point during the sweep, there are s_x local minima of C to the left of π_x, that $C \cap \pi_{-\infty}$ has t components, that the number of splittings and mergings of cross-sectional components to the left of π_x is $Q_x \leq Q$, and that the current value of the count M is M_x. Then, as is easily verified by induction on the sweep events, the total weight of the components of $C \cap \pi_x$ plus M_x is equal to $2Q_x - s_x - t$. Hence, since at $x = +\infty$ the total weight of the components of $C \cap \pi_x$ plus M_x is nonnegative, we have $2Q - s - t \geq 0$, where s is the total number of local minima of C. This implies that $s \leq 2Q$. A symmetric argument applies to the number of local maxima of C, and thus the claim is established.

It therefore remains to estimate the number of critical points of C. For any fixed surface $\sigma \in \Sigma$, the general position assumption is easily seen to imply that there are only $O(1)$ critical points that lie only on σ and on no other surface, so the total number of such points, over all $\sigma \in \Sigma$, is only $O(n)$. The number of critical points that lie on the boundary of one surface of Σ and on a second surface, summed over all pairs of such surfaces, is easily seen to be only $O(n^2)$.

Consider next critical points that lie in the relative interior of an intersection curve $\gamma_{ij} = \sigma_i \cap \sigma_j$, for some pair of surfaces σ_i, $\sigma_j \in \Sigma$ and on no other surface of Σ. If such a point w is a singular point[1] on, say σ_i, then w is an intersection point between the curve of singular points on σ_i and σ_j, and, by the general position assumption, the number of such points, over all pairs σ_i, $\sigma_j \in \Sigma$, is clearly $O(n^2)$. We thus may assume that w is nonsingular on both surfaces, and, by the general position assumption, that σ_i and σ_j meet transversally at w. But then the criticality of w is easily seen to imply that the tangent vector to γ_{ij} at w must be orthogonal to the x-axis, and the number of points on γ_{ij} with this property is $O(1)$ (under the general position assumption). Hence the total number of such points, over all pairs σ_i, $\sigma_j \in \Sigma$, is also $O(n^2)$.

Finally, suppose that w is a critical point that is also a point of intersection of three surfaces σ_i, σ_j, $\sigma_k \in \Sigma$. Arguing as above, we can rule out the case where w is singular on either of these surfaces. Consider now the three intersection curves $\gamma_i = \pi_{x_0} \cap \sigma_i$, $\gamma_j = \pi_{x_0} \cap \sigma_j$, $\gamma_k = \pi_{x_0} \cap \sigma_k$, where x_0 is the x-coordinate of w. These curves meet at w, they are all smooth at w, and $K \cap \pi_{x_0}$ lies on a single side of each of the curves. The number of critical points of this kind, for which any two of these curves are tangent to each other at w, is easily seen, using an argument similar to the one given above, to be only $O(n^2)$. Otherwise, $K \cap \pi_{x_0}$ must be fully contained, locally near w, in just one of the six regions into which these curves split π_{x_0} locally near w. However, this is easily seen to contradict the criticality of w, and thus implies that the total number of critical points of C is $O(n^2)$, as asserted. □

3 Complexity of a Single Cell

We call a vertex v of $\mathcal{A}(\Sigma)$ an *inner vertex* if v is formed by the intersection of the relative interiors of three distinct surfaces of Σ. We concentrate on bounding the number of inner vertices of $C_Z(\Sigma)$, and later on justify the use of this reduced measure of complexity. Our main result is:

Theorem 3.1 *The number of inner vertices of $C_Z(\Sigma)$ is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on ε and on the maximum degree and shape of the surfaces.*

Proof. Let v be an inner vertex of $C = C_Z(\Sigma)$, which is incident to three surfaces σ_1, σ_2 and σ_3. We may assume that v is not a singular point on any of these surfaces, and that these surfaces meet transversally at v. If, say, v is a singular point of σ_1, then it lies on the algebraic curve of bounded degree consisting of all singular points on σ_1. The number of intersection points of this curve with, say σ_2, is constant (under the assumption of general position), which implies that the number of such vertices is only $O(n^2)$. The assumption concerning transversality is also justified by the general position assumption.

For technical reasons, we distinguish between different *sides* of v, adapting the notation of [17] (see also [3, 5, 13]). Formally, the three tangent planes to the surfaces σ_i at v partition 3-space into 8 octants, and a side R of v is any one of these octants. We call the pair (v, R) a *0-border.* We say that (v, R) is a 0-border of C if, when we move from v in any direction that points into R by any sufficiently small distance, we enter C. Our goal is to count the number of inner 0-borders of C, which means that we count each vertex v of C with multiplicity, once for each side of v that lies in C (in the above sense). We define $\kappa(\Sigma)$ to be the number of inner 0-borders on $\partial C_Z(\Sigma)$. We also denote by $\kappa(n)$ the maximum possible value of $\kappa(\Sigma)$, taken over all collections Σ, as above, with a total of n surfaces, and over all cells of $\mathcal{A}(\Sigma)$.

In the foregoing analysis, the notion of a side needs also to be extended to edges and faces bounding C. For an edge e, formed by the intersection of two surfaces σ, σ', we can assume (by reasons similar to those used in the preceding arguments) that no point on e is singular on either of these surfaces, and that σ and σ' cross each other transversally at each point of e. Then at each point $z \in e$, the plane normal to e at z is split by the two tangent planes to σ, σ' at z into 4 quadrants. A

[1] A point v is *singular* on an algebraic surface in $\mathbb{R}^d$, defined by $P = 0$, for some polynomial P, if all the partial derivatives $\partial P / \partial x_i$, for $i = 1, \ldots, d$, vanish at v; see [7, 19].

side of e can be thought of as a *continuous* mapping (in the Hausdorff sense) that maps each point $z \in e$ to one of the quadrants at z (or, rather, to make the Hausdorff continuity well defined, to the intersection of such a quadrant with the unit ball around z). Similarly, a side of a face f can be defined as a continuous mapping from each point $z \in f$ to one of the two unit vectors normal to f at z. If a vertex v is incident to an edge e and lies on another surface σ crossing e transversally, then a side R of v is *consistent* with a side Q of e if the limit of $Q(z)$, as z approaches v, is contained in (the closure of) R, and R is the positive cone spanned by the limit of $Q(z)$ and by the vector normal to σ at z and pointing in the direction of e. Consistency between sides of a vertex and an incident face, or between sides of an edge and an incident face, can be defined in a similar manner. If v is a non-singular vertex of $\mathcal{A}(\Sigma)$, incident to two edges e_1, e_2, which are contained in the same intersection curve of a pair of surfaces, we say that a side R_1 of e_1 is consistent with a side R_2 of e_2 if the two limits of $R_1(z)$ as z approaches v along e_1, and of $R_2(z)$ as z approaches v along e_2, coincide. If R_1', R_2' are the two sides of v consistent with R_1, R_2, respectively, we say that R_2' is the side of v *opposite* to R_1' across the third surface defining v. Given an edge e and a side R of e, we say that (e, R) is a *1-border* of C if, when we move from any point $z \in e$ in a direction contained in $R(z)$, we enter C. Similarly, we can define 2-borders (f, R) of C, for a face f and a side R of f.

Let (v, R) be an inner 0-border of C (non-singular, formed by the transversal intersection of its three incident surfaces, say σ_1, σ_2, σ_3). The corresponding vertex v is incident to (at least) three edges of C, which we denote by e_{12}, e_{13}, and e_{23}, where each e_{ij} is a portion of the corresponding intersection curve $\gamma_{ij} = \sigma_i \cap \sigma_j$, for $1 \le i < j \le 3$; moreover, each edge e_{ij} has a side R_{ij} which is consistent with R, so that (e_{ij}, R_{ij}) is a 1-border of C. If one of these curves, say γ_{12}, contains two edges, e_{12}, e_{12}', with respective sides R_{12}, R_{12}', such that e_{12} and e_{12}' have v as a common endpoint, R_{12} and R_{12}' are consistent with each other, and both (e_{12}, R_{12}) and (e_{12}', R_{12}') are 1-borders of C, then, as is easily seen, there is a face f on σ_3 which is incident to v and which forms with both its sides 2-borders of C. We call such a face a *popular* face of C, borrowing a notation from [3, 5]; see Figure 1. Let us denote by $\pi(\Sigma)$ the maximum number of inner vertices of popular faces bounding a single cell of $\mathcal{A}(\Sigma)$, and let $\pi(n)$ denote the maximum of $\pi(\Sigma)$, over all collections Σ of n surfaces as above (with the same s and b). (Strictly speaking, a vertex v can be incident to more than one popular face, in which case we count it in $\pi(\Sigma)$ with multiplicity, once for every incident popular face.)

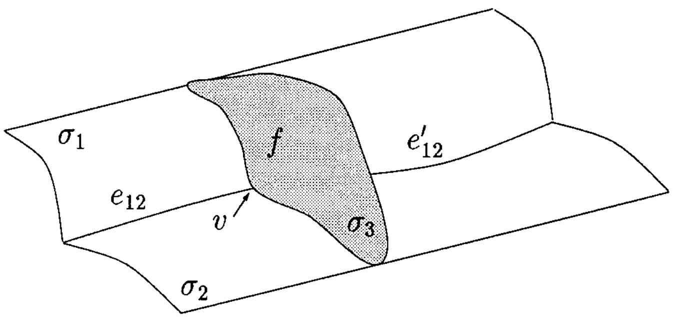


Figure 1: The *popular* face f borders the cell C on both its sides

A major novel ingredient of the proof is the derivation of a sharp upper bound on the number of vertices of popular faces of C; in the previous paper [17] such a bound was derived using special properties of the surfaces that arose in the specific motion planning application that was studied there; here we apply a new technique for obtaining the desired bound in general arrangements. We first obtain an upper bound for the complexity $\kappa(n)$ in terms of the function π, and then proceed to derive an upper bound for $\pi(n)$.

Thus, up to an additive term of $\pi(n)$, it suffices, for the bound on $\kappa(n)$ that we seek, to consider only inner vertices v (or, rather, inner 0-borders (v, R)) which are not incident to any popular face of C.

Let (v, R) be such a 0-border, and let us continue to follow the notations introduced above. For each $1 \le i < j \le 3$, the curve γ_{ij} must contain a maximal open x-monotone connected portion β_{ij} having v as an endpoint, such that the 1-border (β_{ij}, R_{ij}'), where R_{ij}' is the side of β_{ij} consistent with R_{ij}, is disjoint from C. Let z_{ij} denote the other endpoint of β_{ij}.

We define the *index* of v, denoted $j(v)$, to be the number of points of intersections of σ_1, σ_2, σ_3 which lie to the right of v (i.e., with $x > x(v)$). Clearly, $0 \le j(v) \le s - 1$.

We define $\kappa^{(j)}(\Sigma)$, for $j = 0, \ldots, s-1$ to be the maximum number of 0-borders (v, R) of any fixed cell of $\mathcal{A}(\Sigma)$, whose vertices v are inner vertices of index at most j. We also define $\kappa^{(j)}(n)$ to be the maximum possible value for $\kappa^{(j)}(\Sigma)$, over all collections Σ of n surface patches satisfying conditions (i)–(iv) (with the same b and s). Similarly, we define $\pi^{(j)}(\Sigma)$, for $j = 0, \ldots, s-1$, to be the maximum number of vertices with index at most j of all popular faces bounding any fixed cell of $\mathcal{A}(\Sigma)$, where each such vertex is counted with multiplicity, once for every incident popular face. We also define $\pi^{(j)}(n)$ to be the maximum possible value for $\pi^{(j)}(\Sigma)$, over all collections Σ of n surface patches satisfying conditions (i)–(iv) (with the same b and s).

Our method is to derive a recurrence relationship for $\kappa(n)$, by bounding each of the functions $\kappa^{(j)}$ in terms of $\kappa^{(j-1)}$ (with a special handling of $\kappa^{(0)}$); the solution of the resulting system of recurrences will yield the asserted bounds.

So let us fix an index j, and assume that the vertex v under consideration has index at most j. In this case, we incur an overhead of only $\pi^{(j)}(n)$, when we also add to our counts vertices incident to popular faces. Several cases can still arise:

(a) All three arcs β_{ij} emerge from v in the direction of increasing x (or all emerge in the direction of decreasing x). In this case v is an x-extreme vertex of C, as is easily checked (using the general position assumption), and Theorem 2.1 implies that the total number of such vertices (and corresponding 0-borders) is $O(n^2)$. We will thus assume in the sequel that at least one of these arcs emerges from v in the direction of increasing x, and at least one arc emerges in the direction of decreasing x.

(b) At least one of the arcs β_{ij} ends at a point z_{ij} which is either an endpoint of the original intersection curve γ_{ij}, or a point of local x-extremum on that curve. We then charge (v, R) to the point z_{ij}, and note that the number of such points is $O(n^2)$, and that each such point is charged only a constant number of times in this manner (e.g., along β_{ij} it can be charged at most once for every side of β_{ij}), thus implying that the number of 0-borders (v, R) of this kind is only $O(n^2)$. Again, in what follows we assume that this situation does not arise, which means, in particular, that each of the three endpoints z_{ij} is a vertex of C; more precisely, each z_{ij} has a side R^*_{ij} which lies across the third surface defining z_{ij} from a side consistent with R'_{ij}, such that (z_{ij}, R^*_{ij}) is a 0-border of C. See Figure 2.

(c) At least one of the arcs β_{ij}, say for definiteness β_{12}, is not intersected by the third surface σ_3.

Define the *level* of a point w in 3-space to be the smallest number of surfaces of Σ whose removal makes w belong to the closure of the cell containing Z in the resulting subarrangement. If w is a vertex of $\mathcal{A}(\Sigma)$ and R is a side of w, we say that w (resp. (w, R)) lies at *restricted level* $\lambda(w) = k$ (resp. $\lambda((w, R)) = k$) if by removing k surfaces from Σ, none of which is incident to w, we make w a vertex (resp. make (w, R) a 0-border) of the cell containing Z in the resulting subarrangement, and if k is the smallest number with that property.

Let t denote the number of distinct surfaces of Σ that intersect β_{12}. We fix some threshold parameter $\xi = \xi_j$, to be defined later (we will use a different parameter for each j), and consider the following two subcases:

(c.i) $t \geq \xi$: In this case we charge (v, R) to a block of ξ points of intersection between β_{12} and the surfaces of Σ, defined as follows. For each surface σ intersecting β_{12}, choose its point of intersection that lies nearest to v along β_{12}. We obtain at least ξ such designated points, and we charge (v, R) to the block of the first ξ designated points, in their order along β_{12} from v. All those points are inner vertices of $\mathcal{A}(\Sigma)$, and it is clear that none of these vertices can be charged in this manner more than a constant number of times (along β_{12} each such vertex can be charged at most twice for each side of β_{12}). By construction, each of the charged vertices lies at restricted level at most ξ, as is easily verified. Our goal is thus to obtain an upper bound for the number M of inner vertices of $\mathcal{A}(\Sigma)$ that lie at restricted level $\leq \xi$; the number of 0-borders in the present subcase is $O(M/\xi)$.

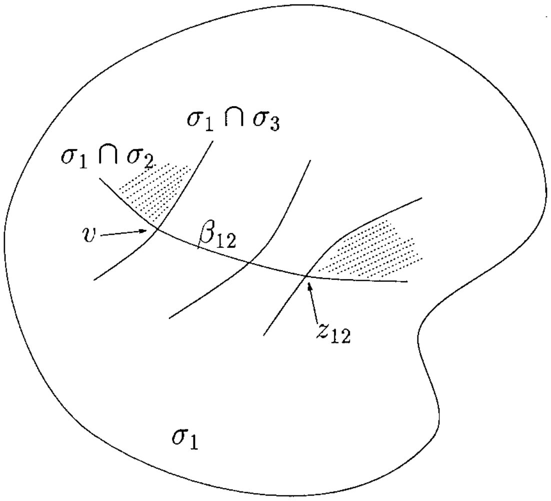


Figure 2: The dotted faces on the surface σ_1 appear on the boundary of C, and near both of them C lies, say, above σ_1

For this we apply the probabilistic analysis technique of [9, 25], in the same manner as in [17]. That is, we choose a random sample $\mathcal{R}$ of $r = n/\xi$ surfaces of Σ, and construct the arrangement $\mathcal{A}(\mathcal{R})$. Let w be an inner vertex of $\mathcal{A}(\Sigma)$ at restricted level $\lambda \leq \xi$, and let Q be a specific collection of λ surfaces, none incident to w, whose removal makes w a vertex of the cell containing Z. The probability that w shows up as a vertex of $C_Z(\mathcal{R})$ is at least $\binom{n-\lambda-3}{r-3}/\binom{n}{r}$: out of the total number $\binom{n}{r}$ of possible samples $\mathcal{R}$, consider those samples that contain the three surfaces forming w and do not contain any of the λ surfaces of Q; for each of these samples (and possibly for other samples as well), w is a vertex of $C_Z(\mathcal{R})$. Hence, we have

$$\sum_{\lambda=0}^{\xi} \frac{\binom{n-\lambda-3}{r-3}}{\binom{n}{r}} F_\lambda \leq E[\kappa(\mathcal{R})] \leq \kappa(r) ,$$

where $E[\cdot]$ denotes expectation (with respect to $\mathcal{R}$), and where F_λ is the number of vertices w of $\mathcal{A}(\Sigma)$ at restricted level λ. (Note that $\kappa(\mathcal{R})$ counts the number of 0-borders bounding $C_Z(\mathcal{R})$, which is clearly an upper bound on the number of vertices of that cell.) As in [9, 25], one easily verifies that, for $r = n/\xi$, we have

$$\sum_{\lambda=0}^{\xi} F_\lambda = O(\xi^3 \kappa(n/\xi)) ;$$

in other words, the number of inner vertices of $\mathcal{A}(\Sigma)$ at restricted level $\le \xi$ is $O(\xi^3\kappa(n/\xi))$, which in turn implies that the number of inner 0-borders (v, R) of C in this subcase is $O(\xi^2\kappa(n/\xi))$.

(c.ii) $t < \xi$: In this case, if we remove these t surfaces from the arrangement, v becomes a vertex of a popular face of C. Indeed, z_{12} has an appropriately consistent side R^*_{12} so that (z_{12}, R^*_{12}) is a 0-border of C. When we remove the t surfaces crossing β_{12}, the cell C expands from R^*_{12} towards v and 'reaches' the other side of v consistent with the side R'_{12} of β_{12}, making v a vertex of a popular face (on σ_3) of the cell containing Z in the reduced arrangement. To exploit this observation, we apply the following variant of the preceding random sampling argument. Fix a parameter $r = n/\xi$, and draw a random sample $\mathcal{R}$ of r surfaces of Σ. Let $E'[\mathcal{R}]$ be the expected number of vertices in $\mathcal{A}(\mathcal{R})$ of index $\le j$ which are incident to popular faces of $C_Z(\mathcal{R})$ (counted with the appropriate multiplicity). By definition, $E'[\mathcal{R}] \le \pi^{(j)}(r)$. (Note that the index of a vertex does not change when we pass to a subset $\mathcal{R}$, as long as the 3 surfaces defining the vertex belong to $\mathcal{R}$.) Now, using a similar argument to the one given above, the probability that our vertex v will show up as a vertex of such a popular face of $C_Z(\mathcal{R})$ is $\ge \binom{n-t-3}{r-3}/\binom{n}{r}$: of the $\binom{n}{r}$ possible ways of choosing $\mathcal{R}$, we consider those samples for which $\sigma_1, \sigma_2, \sigma_3 \in \mathcal{R}$, and none of the other t surfaces crossing β_{12} is chosen in $\mathcal{R}$; as already noted, each such choice (and possibly others as well) will make v appear as a vertex of a popular face of the cell under consideration. Hence, we have

$$\sum_{t=0}^{\xi} \frac{\binom{n-t-3}{r-3}}{\binom{n}{r}} G_t \le E'[\mathcal{R}] \le \pi^{(j)}(r) ,$$

where G_t is the number of 0-borders (v, R) in the full arrangement that fall into the present subcase, with $j(v) \le j$ and with exactly t surfaces crossing the corresponding arc β_{12}. Arguing exactly as above, we obtain, for $r = n/\xi$,

$$\sum_{t=0}^{\xi} G_t = O(\xi^3 \pi^{(j)}(n/\xi)) ;$$

in other words, the number of 0-borders in this subcase is $O(\xi^3\pi^{(j)}(n/\xi))$.

(d) In the remaining case (which can occur only if $j(v) > 0$), the third surface σ_3 intersects β_{12} in at least one point w; if there are several such points (no more than j by assumption), we take w to be the point lying furthest from v along β_{12}. Let t denote the number of distinct surfaces of Σ, *excluding* σ_3, that intersect β_{12}. We consider the following two subcases:

(d.i) $t \ge \xi$: In this case we charge (v, R) to a block of ξ vertices of the full arrangement $\mathcal{A}(\Sigma)$ which lie along β_{12}, in complete analogy to the construction in case (c.i) above, except that the surface σ_3 is excluded from the construction. Since each such vertex can be charged in this manner only a constant number of times, and all these vertices lie at restricted level $\le \xi + 1$, as is easily checked, it follows, exactly as above, that the total number of 0-borders v in this subcase is $O(\xi^2\kappa(n/\xi))$.

(d.ii) $t < \xi$: In this case, if we remove these t surfaces (without removing σ_3), the point w, together with a side R_w consistent with R'_{12}, must form a 0-border (w, R_w) of the cell containing Z in the reduced arrangement. Indeed, recall that we are assuming that the other endpoint z_{12} of β_{12} forms a 0-border (z_{12}, R^*_{12}) of C, for a side R^*_{12} lying across from a side consistent with R'_{12}. By assumption, the portion of β_{12} between w and z_{12} is not crossed by σ_3, so, when the other t surfaces crossing β_{12} are removed, the cell C expands from the side R^*_{12} and 'reaches' w from z_{12} along the side (β_{12}, R'_{12}). We charge (v, R) to (w, R_w). Clearly, each such (w, R_w) is charged in this manner only a constant number of times.

We next estimate the number of 0-borders (w, R_w) of this kind. We apply a random-sampling argument similar to those used above. That is, we fix a parameter $r = n/\xi$, and draw a random sample $\mathcal{R}$ of r surfaces of Σ. Let $E''[\mathcal{R}]$ be the expected number of 0-borders (w, R_w) of $C_Z(\mathcal{R})$, such that w has index $\le j - 1$. By definition, $E''[\mathcal{R}] = E[\kappa^{(j-1)}(\mathcal{R})] \le \kappa^{(j-1)}(r)$.

The probability that the charged 0-border (w, R_w) will show up as such a 0-border of $C_Z(\mathcal{R})$ is $\ge \binom{n-t-3}{r-3}/\binom{n}{r}$: of the $\binom{n}{r}$ possible ways of choosing $\mathcal{R}$, we consider those samples for which $\sigma_1, \sigma_2, \sigma_3 \in \mathcal{R}$, and none of the other t surfaces crossing β_{12} is chosen in $\mathcal{R}$; each such choice (and possibly other choices too) will make (w, R_w) appear as a 0-border of $C_Z(\mathcal{R})$, as argued above. Hence, we have

$$\sum_{t=0}^{\xi} \frac{\binom{n-t-3}{r-3}}{\binom{n}{r}} H_t = O(E''[\mathcal{R}]) = O(\kappa^{(j-1)}(r)) ,$$

where H_t is the number of 0-borders (v, R) in the full arrangement that fall in the present subcase, with $j(v) \le j$ and with exactly t surfaces crossing the corresponding arc β_{12} (excluding the corresponding surface σ_3). Arguing as above, we obtain, for $r = n/\xi$,

$$\sum_{t=0}^{\xi} H_t = O(\xi^3 \kappa^{(j-1)}(n/\xi)) ;$$

in other words, the number of 0-borders (v, R) in this subcase is $O(\xi^3\kappa^{(j-1)}(n/\xi))$.

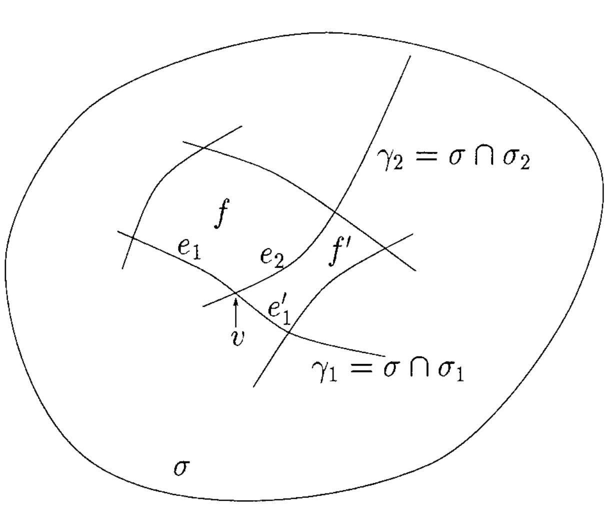

$\gamma_2 = \sigma \cap \sigma_2$
f
e_1
e_2
f'
e_1'
v
$\gamma_1 = \sigma \cap \sigma_1$
σ

Figure 3: Two adjacent popular faces f, f', giving rise to a popular edge e_2

Hence, summing over all cases, we obtain the following recurrence for $\kappa^{(j)}$ (where, for $j = 0$, we put $\kappa^{(-1)} = 0$ in the right-hand side):

$$\kappa^{(j)}(n) = O(\xi^2\kappa(n/\xi) + \xi^3\pi^{(j)}(n/\xi) + \xi^3\kappa^{(j-1)}(n/\xi) + n^2 + \pi^{(j)}(n)) \; . \qquad (1)$$

In order to solve the recurrence (1) we first have to bound the functions $\pi^{(j)}(n)$, which is done in the next subsection.

3.1 The Number of Vertices on Popular Faces

To bound the number of such vertices, we adapt the analysis given above to bound the functions $\pi^{(j)}(n)$, rather than the functions $\kappa^{(j)}(n)$.

Let v be an inner vertex of a popular face f of the cell C; assume that f lies on a surface $\sigma \in \Sigma$, and that the two other surfaces incident to v are $\sigma_1, \sigma_2 \in \Sigma$. As above, we may assume that v is non-singular on any of these surfaces, and that these surfaces meet transversally at v. Denote the two sides of f by R^+ and R^-. Let $\gamma_i = \sigma_i \cap \sigma$, for $i = 1, 2$. Each γ_i contains an edge e_i having v as an endpoint and bounding f. Let R_i^+, R_i^- denote the two sides of e_i that are consistent with R^+, R^-, respectively.

If, say γ_1 has another edge e_1' incident to v such that (a) e_1' bounds another popular face f' on σ, and (b) f and f' share the edge e_2, then e_2 is a *popular edge* of C, meaning that all its four sides lie in C locally near e_2 (see Figure 3). We claim that the number of popular edges of C is $O(n^2)$. This follows from the observation that v must be a locally x-extreme vertex of one of these four sides[2] (assuming general position), and, by Theorem 2.1, the number of such vertices, and hence also the number of popular edges, is $O(n^2)$. Moreover, if both e_1 and e_2 emanate from v in the positive x-direction, or if both emanate in the negative x-direction, then, as is easily checked, v must be a locally x-extreme vertex of one of the two sides of f, so the number of such vertices is also $O(n^2)$. Hence, in what follows we may assume that neither of the edges e_1, e_2 is adjacent along σ to another popular face of C, and that one of these edges emanates from v in the positive x-direction and one emanates in the negative x-direction.

As above, our method is to derive a recurrence relationship for $\pi(n)$, by bounding each of the functions $\pi^{(j)}$ in terms of $\pi^{(j-1)}$ (with a special handling of $\pi^{(0)}$); the solution of the resulting system of recurrences will yield the asserted bounds.

So let us fix an index j, and assume that the vertex v under consideration has index $\leq j$. By our assumptions, for $i = 1, 2$, the curve γ_i must contain a maximal open x-monotone connected portion β_i having v as an endpoint and satisfying the following property: let $R_i'^+$, $R_i'^-$ be the two sides of β_i which are consistent with the two respective sides R_i^+, R_i^- of e_i; then β_i does not contain any point at which both sides $R_i'^+$, $R_i'^-$ lie locally in C. Let z_i denote the other endpoint of β_i. By assumption, one of these arcs, say β_1, emanates from v in the positive x-direction and the other emanates in the negative x-direction.

Several cases can arise:

(a) At least one of the arcs β_i ends at a point z_i which is either an endpoint of the original intersection curve γ_i, or a point of local x-extremum on that curve. We then charge v to the point z_i, and note that the number of such points is $O(n^2)$, and that each such point is charged only a constant number of times in this manner, thus implying that the number of vertices v of this kind is only $O(n^2)$. In what follows we assume that this situation does not arise, which means, in particular, that z_1 is a vertex of another popular face of C, whose two sides are consistent with (appropriate extensions along γ_1 of) $R_1'^+$, $R_1'^-$, respectively.

In the full version of the paper we distinguish and analyze two additional cases: **(b)** β_1 is not intersected by the third surface σ_2, and **(c)** the surface σ_2 intersects β_1 in at least one point (which can occur only when $j(v) > 0$). Due to lack of space, we omit further details for these cases here, and we just summarize with the resulting recurrence for $\pi^{(j)}$ (where, for $j = 0$, we put $\pi^{(-1)} = 0$ in the right-hand side):

$$\pi^{(j)}(n) = O(\xi^2\pi(n/\xi) + \xi n^2 + \xi^3\pi^{(j-1)}(n/\xi)) \; . \qquad (2)$$

[2] Strictly speaking, v is locally x-extreme in one of the four portions of C into which these sides 'point'; we allow ourselves here and below this slight abuse of notation.

3.2 Solving the Recurrences

We next proceed to solve the recurrences (1) and (2). We start with the latter equation, fix some $\varepsilon > 0$, and claim that its solution is $\pi^{(j)}(n) \leq B_j n^{2+\varepsilon}$, for $j = 0, \ldots, s-1$, where the constants B_j depend on ε, j, and on the maximum degree b. By definition, this implies that $\pi(n) \leq B_{s-1} n^{2+\varepsilon}$.

We prove this claim by induction on n. We first rewrite (2), using a different parameter ξ_j for each j, as

$$\pi^{(0)}(n) \leq c\xi_0^2 \pi(n/\xi_0) + c\xi_0 n^2 \qquad \text{and} \qquad (3)$$
$$\pi^{(j)}(n) \leq c\xi_j^2 \pi(n/\xi_j) + c\xi_j n^2 + c\xi_j^3 \pi^{(j-1)}(n/\xi_j) \ , \quad j = 1, \ldots, s-1,$$

for appropriate positive constants c, $\xi_0, \ldots, \xi_{s-1}$; without loss of generality, we assume $c > 1$. We take ξ_0 to be sufficiently large, and put $\xi_j = \xi_0^{\varepsilon^j}$, for $j = 0, \ldots, s-1$; note that $\xi_j = \xi_{j-1}^{\varepsilon}$, for $j = 1, \ldots, s-1$. We note that, by choosing the B_j's to be sufficiently large, we can assume that the claimed bounds hold for all $n \leq \xi_0$. (E.g., choose the B_j's to be larger than some appropriate multiple of ξ_0, and use the fact that all the quantities we want to estimate are bounded by $O(n^3)$, where the constant of proportionality depends only on the maximum degree b of the surfaces and of their boundaries.)

For $n > \xi_0$, we apply the induction hypothesis in the right-hand side of (3), and conclude that the asserted bounds continue to hold for n too, provided that the following inequalities are satisfied:

$$\frac{cB_{s-1}}{\xi_0^{\varepsilon}} + c\xi_0^{1-\varepsilon} \leq B_0 \ , \qquad \text{and}$$
$$\frac{cB_{s-1}}{\xi_j^{\varepsilon}} + \frac{c\xi_j}{\xi_0^{\varepsilon}} + c\xi_j^{1-\varepsilon} B_{j-1} \leq B_j \ , \quad j = 1, \ldots, s-1.$$

To achieve this, we choose $B_0 > 2c\xi_0^{1-\varepsilon}$, and $B_j = \frac{(j+1)c^j}{\xi_j^{\varepsilon}} \xi_0^{\varepsilon} B_0$, and require that the ξ_j's be sufficiently large so that $\xi_0^{\varepsilon^s} > 2sc^s$. We leave it to the reader to verify that this choice of coefficients satisfies the above inequalities. This inductive step completes the solution of the recurrence for the functions $\pi^{(j)}(n)$.

In a similar manner we solve the recurrences (1). We show that the solution of these equations is $\kappa^{(j)}(n) \leq A_j n^{2+\varepsilon}$, for $j = 0, \ldots, s-1$, where the constants A_j depend on ε, j, and the maximum degree b. By definition, this implies that $\kappa(n) \leq A_{s-1} n^{2+\varepsilon}$. For lack of space, we omit the fairly routine inductive proof of these bounds.

Using rather standard arguments (given in the full version of this paper), one can show that the overall combinatorial complexity of C is dominated asymptotically by the above bound for the number of its inner vertices. We thus obtain the main result of the paper:

Theorem 3.2 *The combinatorial complexity of a single cell in an arrangement of n algebraic surface patches in 3-space, satisfying the conditions (i)–(iv), is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on ε, s and b.*

4 Zone Complexity

An interesting application of Theorem 3.2 is to bound the combinatorial complexity of the *zone* of a surface in an arrangement of other surfaces in 3-space. Specifically, let Σ be a collection of n algebraic surface patches in 3-space, and let σ be another such surface, so that the surfaces in $\Sigma \cup \{\sigma\}$ satisfy conditions (i)–(iv). The zone of σ in $\mathcal{A}(\Sigma)$ is the collection of all cells of $\mathcal{A}(\Sigma)$ that are crossed by σ. The complexity of the zone is the sum of the complexities of all its cells.

Theorem 4.1 *The combinatorial complexity of the zone of σ in $\mathcal{A}(\Sigma)$ is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on ε, s and b.*

Proof. We extend the idea used in [12] for the analysis of zones in 2-dimensional arrangements. That is, assume first that σ is a connected surface. We cut each $\sigma_i \in \Sigma$ into a constant number of sub-patches along its curve of intersection with σ. If we shrink these pieces by a small amount, all the cells of the zone become connected to each other, and form a single cell in the arrangement of the new patches. Since these patches are easily seen to also satisfy conditions similar to (i)–(iv), the asserted bound follows. If σ is not connected, it consists of a constant number of connected components, and we apply the above argument to each component separately. □

5 Conclusion

In this paper we have obtained a near-quadratic bound for the combinatorial complexity of a single cell in an arrangement of n algebraic surface patches of constant maximum degree, each bounded by a constant number of algebraic arcs of constant maximum degree as well. This almost settles a long-standing conjecture, and provides a fairly satisfactory extension of the 2-dimensional Davenport-Schinzel theory developed in [14]. Our proof extends recent techniques developed in [16, 17, 26], all based on the probabilistic technique of [9, 25] for deriving bounds on the number of generalized '($\leq k$)-sets' in arrangements.

As noted in the introduction, our result has immediate applications to the general motion planning problem with three degrees of freedom. That is, for rather general systems with three degrees of freedom, the combinatorial complexity of the connected component of the free configuration space, consisting of all robot placements which are reachable from some given initial free placement, is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$, where n is the number of contact surfaces, as defined in the introduction. This still falls short of efficient construction of such a component (in near-quadratic running time, which is one of the major open problems that we pose in this paper). However, at least we know that the complexity of such a component is nearly an order of magnitude smaller than the worst-case complexity of the entire arrangement, and, in most cases, also of the entire free configuration space.

Another open problem that the paper raises is to further improve the bound that we have obtained, to the conjectured bound of $O(n\lambda_s(n))$, or at least to $O(n^2\text{polylog}(n))$. Another open problem is to extend our results to arrangements in d dimensions. We believe that this is doable, and are currently exploring this problem. One main subproblem here is to extend Theorem 2.1 to higher dimensions.

Other open problems are to extend our analysis to obtain sharp bounds on the complexity of many cells in 3-D arrangements, and to derive bounds on the sum of squares of cell complexities in an entire arrangement; see [2, 5] for related work.

References

[1] P.K. Agarwal, M. Sharir and P. Shor, Sharp upper and lower bounds for the length of general Davenport Schinzel sequences, *J. Combin. Theory, Ser. A.* 52 (1989), 228–274.

[2] B. Aronov, J. Matoušek and M. Sharir, On the sum of squares of cell complexities in hyperplane arrangements, *Proc. 7th Symp. on Computational Geometry* (1991), pp. 307–313. (To appear in *J. Comb. Theory Ser. A.*)

[3] B. Aronov, M. Pellegrini and M. Sharir, On the zone of a surface in a hyperplane arrangement, *Discrete Comput. Geom.* 9 (1993), 177–186.

[4] B. Aronov and M. Sharir, Triangles in space, or building (and analyzing) castles in the air, *Combinatorica* 10 (1990), 137–173.

[5] B. Aronov and M. Sharir, Castles in the air revisited, *Proc. 8th ACM Symp. on Computational Geometry* (1992), 146–156. (To appear in *Discrete Comput. Geom.*)

[6] B. Aronov and M. Sharir, Translational motion planning of a convex polyhedron in 3 dimensions, *this volume.*

[7] J. Bochnak, M. Coste and M-F. Roy, *Géométrie Algébrique Réelle*, Springer-Verlag, Berlin 1987.

[8] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir and J. Snoeyink, Computing a single face in an arrangement of line segments and related problems, *SIAM J. Computing* 22 (1993) (in press).

[9] K. Clarkson and P. Shor, Applications of random sampling in computational geometry, II, *Discrete Comput. Geom.* 4 (1989), 387–421.

[10] M. de Berg, L.J. Guibas and D. Halperin, Vertical decompositions for triangles in 3-space, *this volume.*

[11] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[12] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel and M. Sharir, Arrangements of curves in the plane – Topology, combinatorics, and algorithms, *Theoretical Computer Science* 92 (1992), 319–336.

[13] H. Edelsbrunner, R. Seidel and M. Sharir, On the zone theorem in hyperplane arrangements, *SIAM J. Computing* 22 (1993), 418–429.

[14] L.J. Guibas, M. Sharir and S. Sifrony, On the general motion planning problem with two degrees of freedom, *Discrete and Comput. Geom.* 4 (1989), 491–521.

[15] D. Halperin, *Algorithmic Motion Planning via Arrangements of Curves and of Surfaces,* Ph.D. Dissertation, Computer Science Department, Tel Aviv University, July 1992.

[16] D. Halperin and M. Sharir, New bounds for lower envelopes in three dimensions, with applications to visibility in terrains, *Proc. 9th ACM Symp. on Computational Geometry* (1993), pp. 11–18. Also to appear in *Discrete Comput. Geom.*

[17] D. Halperin and M. Sharir, Near-quadratic bounds for the motion planning problem for a polygon in a polygonal environment, *Proc. 34th IEEE Symp. on Foundations of Computer Science* (1993), pp. 382–391.

[18] S. Hart and M. Sharir, Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes, *Combinatorica* 6 (1986), 151–177.

[19] R. Hartshorne, *Algebraic Geometry*, Springer-Verlag, New York 1977.

[20] M.W. Hirsch, *Differential Topology*, Springer-Verlag, New York 1976.

[21] P. McMullen, The maximum number of faces of a convex polytope, *Mathematika* 17 (1970), 179–184.

[22] J. Pach and M. Sharir, The upper envelope of piecewise linear functions and the boundary of a region enclosed by convex plates: Combinatorial analysis, *Discrete Comput. Geom.* 4 (1989), 291–309.

[23] R. Pollack and M.F. Roy, On the number of cells defined by a set of polynomials, *C.R. Acad. Sci. Paris*, t. 316, Série I (1993), 573–577.

[24] J.T. Schwartz and M. Sharir, On the two-dimensional Davenport Schinzel problem, *J. Symbolic Computation* 10 (1990), 371–393.

[25] M. Sharir, On k-sets in arrangements of curves and surfaces, *Discrete Comput. Geom.* 6 (1991), 593–613.

[26] M. Sharir, Almost tight upper bounds for lower envelopes in higher dimensions, *Proc. 34th IEEE Symp. on Foundations of Computer Science* (1993), pp. 498–507. Also to appear in *Discrete Comput. Geom.*

On Translational Motion Planning in 3-Space*

Boris Aronov[†] Micha Sharir[‡]

Abstract

Let B be a convex polyhedron translating in 3-space amidst k convex polyhedral obstacles $A_1, \ldots, A_k$ with pairwise disjoint interiors. The *free configuration space* (space of all collision-free placements) of B can be represented as the complement of the union of the Minkowski sums $P_i = A_i \oplus (-B)$, for $i = 1, \ldots, k$. We show that the combinatorial complexity of the free configuration space of B is $O(nk \log^2 k)$, where n is the total complexity of the individual Minkowski sums $P_1, \ldots, P_k$. The bound is almost tight in the worst case. We also derive an efficient randomized algorithm that constructs this configuration space in expected time $O(nk \log^3 k)$.

1 Introduction

Let $A_1, \ldots, A_k$ be k closed convex polyhedra in three dimensions with pairwise disjoint interiors, and let B be another closed convex polyhedron, which, with no loss of generality, is assumed to contain the origin O. In the context of motion planning, B is a 'robot' free to only translate in 3-space, and the A_i's are obstacles which B must avoid. Suppose A_i has q_i faces, B has p faces, and put $q = \sum_{i=1}^{k} q_i$. Let

$$P_i = A_i \oplus (-B) = \{a - b \mid a \in A_i,\, b \in B\}$$

be the *Minkowski sum* of A_i and $-B$, for $i = 1, \ldots, k$, let $\mathcal{P} = \{P_i\}_{i=1}^{k}$ be the resulting collection of these so-called *expanded obstacles*, and let $U = \bigcup_{i=1}^{k} P_i$ be their union. As is well known, the complement C of U (also called the *common exterior* of $\mathcal{P}$) represents the *free configuration space* FP of B, in the sense that, for each point $z \in C$, the placement of B, for which the reference point O lies at z, does not intersect any of the obstacles A_i, and all such free placements are represented in this manner.

As is well known, the combinatorial complexity (i.e., the number of vertices, edges, and faces on the boundary) of each P_i is at most $\Theta(pq_i)$, so the sum of the complexities of the expanded obstacles is $n = O(pq)$. In practice, n is usually much smaller than pq. The bounds derived in this paper depend only on n and k, and not on p and q.

Our main result is that the combinatorial complexity of the union U, and thus also of $FP = C$, is $O(nk \log^2 k)$ and $\Omega(nk\alpha(k))$ in the worst case. This should be compared to the recent bound obtained by the authors [4] on the combinatorial complexity of the union of any k convex polyhedra in 3-space with a total of n faces; it is shown there that the maximum complexity of such a union is $O(k^3 + nk \log^2 k)$ and $\Omega(k^3 + nk\alpha(k))$. Thus the convex polyhedra P_i arising in the context of translational motion planning have special properties that yield the above improved bound.

The problem of obtaining sharp bounds for the combinatorial complexity of the union of Minkowski sums, as above, has been open for the past eight years, and has been studied during this period in several papers. It was raised by Kedem *et al.* [16] (see also [20]), where the 2-dimensional version of the problem was successfully tackled. Related work on the 3-dimensional case is described in [2, 3, 4, 10, 14, 19]. Except for the papers [4, 14] (and parts of [2]), these papers studied different (though related) problems, involving the complexity of the lower envelope or of a single cell in an

[*] Work on this paper by the first author has been supported by NSF Grant CCR-92-11541. Work by the second author has been supported by NSF grant CCR-91-22103, by a Max Planck Research Award, and by grants from the U.S.-Israeli Binational Science Foundation, the G.I.F. — the German-Israeli Foundation for Scientific Research and Development, and the Fund for Basic Research administered by the Israeli Academy of Sciences.

[†] Department of Computer Science, Polytechnic University, Brooklyn, NY 11201 USA.

[‡] School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 USA.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

arrangement of triangles in 3-space. A recent paper of Halperin and Yap [14] analyzes the complexity of the union of Minkowski expansions, as above, for the case where B is a box, and obtains a bound of $O(n^2\alpha(n))$. Our result is general, and thus almost settles this open problem, except for the polylogarithmic factor in our upper bound; we conjecture that the correct bound is $O(nk\alpha(k))$.

We also consider the algorithmic problem of efficient construction of FP. We present an efficient and rather simple randomized algorithm that computes FP in expected time $O(nk\log^3 k)$. The algorithm and its analysis are adapted from a very similar algorithm given in [4] for constructing the union of arbitrary convex polyhedra in 3-space.

The paper is organized as follows. In Section 2 we bound the number of components and local minima of C. The analysis of the topology of U, and of several related structures constructed from a family of convex polyhedra, continues in Section 3. Section 4 establishes the main result of the paper, namely the bounds on the combinatorial complexity of U. Section 5 describes the randomized algorithm for computing the boundary of U and Section 6 concludes with some remarks and open problems.

2 The Number of Holes of U and Local Extrema of C

We first simplify the analysis by assuming that the given polyhedra A_i and B are in *general position*, meaning that the coordinates of the vertices of the A_i's and of B are all transcendentals that are algebraically independent over the rationals. As argued in [4], the general position assumption involves no real loss of generality. Without loss of generality, in what follows we will also assume that U is bounded and that C has thus a single unbounded component.

To establish our upper bound on the complexity of the union U, it clearly suffices to bound the number of vertices of U. Moreover, it suffices to bound the number of vertices of U which are formed by the intersection of the relative interiors of three faces of three distinct polyhedra P_i, because the number of all other vertices, namely, vertices of the original P_i's and intersections of edges of the original P_i's with faces of other expanded obstacles, is only $O(nk)$. This is a consequence of the following easy proposition (cf. [4]):

Proposition 2.1 *The number of vertices of $P_i \cap P_j$, summed over all $P_i, P_j \in \mathcal{P}$, is $O(nk)$.*

We first bound the number of connected components of the complement C of U (the common exterior of $\mathcal{P}$). Under the assumptions made above, C has a unique unbounded component, so it suffices to estimate the number of bounded components of C, i.e., "holes" in U.

Let an *intersection edge* be an edge of the intersection of the boundaries of two distinct expanded obstacles. Each such edge e meets the boundary of the union U in at most $k-1$ intervals, as it intersects each of the remaining $k-2$ polyhedra in at most one interval, and the complement (within e) of the union of these $k-2$ intervals consists of at most $k-1$ intervals.

We introduce a real parameter $t \in [0,1]$ (we refer to it as 'time'), define $P_i(t) = A_i \oplus (-tB)$, for $i = 1,\ldots,k$, and $U(t) = \bigcup_{i=1}^k P_i(t)$, and let $C(t)$ denote the complement of $U(t)$. Note that the general position assumption does not imply that polyhedra $\{P_i(t)\}$ are in general position, for all values of t. It does imply, though, that, at any given t, we can have at most one degenerate contact between the $P_i(t)$'s. Such a contact can be expressed as a polynomial equality in t and in the coordinates of the vertices of the A_i's and of B. Moreover, there is only a finite number of values of t at which such a degeneracy can occur. For example, if $t_0 > 0$ is the first instant at which $P_i(t)$ and $P_j(t)$ meet, then $P_i(t) \cap P_j(t)$ must consist of a single point, for otherwise we would have at least two algebraically-independent polynomial equalities holding simultaneously, which is forbidden by the general position assumptions. Moreover, such an initial single-point contact of $P_i(t)$ and $P_j(t)$ must occur either between a vertex of one of these polyhedra and a face of the other, or between two edges, one from each polyhedron. Similarly, it is easy to verify that the first contact between three growing obstacles must occur at a single point.

Before continuing, we recall a well-known fact: Let P and Q be two convex polyhedra. Denote by $\mathcal{N}_P$ the *normal diagram* (also known as the *Gaussian diagram*) of P; this is the decomposition of the Gaussian sphere $\mathcal{S}^2$ of orientations into regions, each consisting of the outward-directed normal directions of all planes supporting P at some vertex (each edge of $\mathcal{N}_P$ is a great circular arc consisting of the outward-directed normal directions of all planes supporting P at some edge, and each vertex of $\mathcal{N}_P$ is the outward-directed normal direction of some face of P). The normal diagram $\mathcal{N}_Q$ of Q is defined analogously. Then the overlay of $\mathcal{N}_P$ and $\mathcal{N}_Q$ is the normal diagram of the Minkowski sum $P \oplus Q$. This implies that the combinatorial structure (i.e., the number of, and incidence relations between, faces of all dimensions) of $P \oplus Q$ is completely determined by the orientations of the faces of P and of Q and by their incidence structures. Hence, in particular, the combinatorial structure of $P_i(t) = A_i \oplus (-tB)$ is the same for all $t > 0$. This implies the following property. For $t \in [0,1]$, let $s(t)$ denote the Minkowski sum either of a fixed edge of A_i and $(-t)$ times a fixed vertex of B, or of a fixed vertex of A_i and $(-t)$ times a fixed edge of B. If $s(t_0)$ is an edge of $P_i(t_0)$, for some $t_0 \in (0,1]$, then $s(t)$ is an edge of $P_i(t)$ for all $t \in (0,1]$.

We want to keep track of the number of holes of $U(t)$ as t increases from 0 to 1. At $t = 0$, the number of holes of $U(t)$ is 0. As t increases, the holes shrink, so the number of holes can *increase* only when a component of $C(t)$ is split into two components.

Lemma 2.2 *A component of $C(t)$ can split into two components only at times t when an edge $s(t)$ of some $P_\ell(t)$ meets an intersection edge $e(t)$ of two other polyhedra, $P_i(t)$, $P_j(t)$, at some point v, so that the intersection of $C(t)$ with a small neighborhood of v is disconnected.*

Proof: Easily follows from an observation of de Berg *et al.* [5] that a split can occur only when some $P_i(t)$, $P_j(t)$, $P_\ell(t)$ come to intersect for the first time, and from general position assumption. We omit the details. □

Suppose that such a configuration does indeed arise at some time t, and suppose that $e(t)$ is formed by the intersection of a pair of faces of $P_i(t)$ and of $P_j(t)$. The edge $s(t)$ intersects both $P_i(t)$ and $P_j(t)$ at two respective intervals having a common endpoint. Lemma 2.2 then implies that the two intervals $s(t) \cap P_i(t)$ and $s(t) \cap P_j(t)$ must have disjoint relative interiors.

Lemma 2.3 *If at time t the intervals $s(t) \cap P_i(t)$ and $s(t) \cap P_j(t)$ intersect, then $s(t') \cap P_i(t')$ and $s(t') \cap P_j(t')$ also intersect, for all $t' > t$.*

Proof: Omitted in this version.

Lemmas 2.2 and 2.3 imply that, for each t at which a connected component of C is split into two components, there exist two polyhedra, $P_i(t)$, $P_j(t)$, and an edge $s(t)$ of another polyhedron $P_\ell(t)$, such that the two intervals $s(t) \cap P_i(t)$ and $s(t) \cap P_j(t)$ meet at a common endpoint, and continue to overlap for all $t' > t$. Thus the number of such 'hole-splitting' events, involving the same edge $s(t)$ of any of the expanded polyhedra, is at most $k - 3$, so the total number of holes that the union can have is at most $O(nk)$. That is, we have shown:

Theorem 2.4 *The number of connected components of the common exterior of Minkowski sums of k polyhedra, as above, is $O(nk)$.*

Remark: We do not know if this bound is tight, as the best lower bound we can construct is $\Omega(k^2)$.

Let S be any closed polyhedral set in 3-space in general position. (In this paper we only consider polyhedral sets bounded by a finite number of edges, vertices, and faces.) For a point p, let a *neighborhood* of p in S be the intersection of S with a ball centered at p, whose radius is smaller than the distance from p to any edge, face, or vertex of S not incident to p. A point p of S is a *local minimum* if it is a point of smallest z-coordinate in some neighborhood of p in S. (Under the general position assumptions, no face or edge can be parallel to the xy-plane, so p will be the only point in its neighborhood with this z-coordinate.) A *local maximum* is defined similarly. The following theorem is a consequence of Theorem 2.4:

Theorem 2.5 *The number of local maxima and minima of the closure $\bar{C}$ of C is $O(nk)$.*

Proof: We prove the claim only for the number of local minima of $\bar{C}$, since the treatment of local maxima is fully symmetric. Any bounded convex component K of $\bar{C}$ has exactly one local minimum (which is also the global minimum of K), because K does not have a lowest horizontal edge, as implied by our general position assumption. Hence the number of such minima is $O(nk)$, by Theorem 2.4. This also applies to components of $\bar{C}$ all of whose horizontal cross sections are connected, because such a component has at most one local (and global) minimum as well. So we will proceed to bound the number of local minima of those components that have at least one disconnected horizontal cross section. We will in fact prove that the number of such minima is only $O(k^2)$.

Sweep $\bar{C}$ with a plane π parallel to the xy-plane and moving upwards, in the positive z-direction, and keep track of the number I of connected components of $\bar{C} \cap \pi$. This number is initially 1 (when π is below all vertices of $\bar{C}$). I increases by 1 when π sweeps through a local minimum of $\bar{C}$, or when a connected component of $\bar{C} \cap \pi$ splits into two subcomponents; I decreases by 1 when π sweeps through a local maximum of $\bar{C}$, or when two components of $\bar{C} \cap \pi$ merge into a single component. The events at which components may split or merge occur only when π sweeps through the topmost or bottommost vertex of the intersection of some pair P_i, P_j of expanded obstacles, for some $i \neq j$. The number of such events is clearly bounded by $2\binom{k}{2} = O(k^2)$.

Consider the following dynamic scheme for assigning weights to each component of $\bar{C} \cap \pi$. When π sweeps through a local minimum point of $\bar{C}$, a new component of $\bar{C} \cap \pi$ is created, and is assigned weight -1. When two components of $\bar{C} \cap \pi$ merge, the weight assigned to the new component is 2 plus the sum of the weights of the merged components. When a component shrinks and disappears, its final weight is added to a global count. When a component is split into two subcomponents, each of them is assigned weight $1 + \frac{w}{2}$, where w is the weight of the split component.

We claim that, at any given time during the sweep, the weight of any component of $\bar{C} \cap \pi$ is always at least -1, and the weight of a component that was formed by a splitting or merging operation is non-negative. (We refer to components of $\bar{C}$ that have a cross-sectional component of the latter type as *interesting components.*) Both claims are easy to prove by induction on the sweep events. Now suppose that, at some point during the sweep, there are s local minima of $\bar{C}$ below π, so that the cross-sectional components that they have created have already been split or merged, and that the number of such splittings and mergings below π is N (which is at most $O(k^2)$). Then the total weight of the components of $\bar{C} \cap \pi$ that have been formed by splittings and mergings, plus the total weight of all the interesting components of $\bar{C}$ that have terminated below π, is easily seen, by induction on the sweep events, to be $2N - s$. As claimed above, this weight is non-negative, thus $s \leq 2N = O(k^2)$. When the sweep plane passes the topmost vertex of $\bar{C}$, this inequality shows that the number of local minima of $\bar{C}$ contained in interesting components is $O(k^2)$. Adding the number of local minima in non-interesting components of $\bar{C}$, we obtain the bound asserted in the theorem. □

3 On the Structure of Levels in $\mathcal{A}(\mathcal{P})$

Define the *arrangement* $\mathcal{A}(\mathcal{P})$ of the collection $\mathcal{P}$ of the expanded obstacles to be the decomposition of space into vertices, edges, faces, and 3-dimensional cells, induced by the faces of the polyhedra of $\mathcal{P}$; for more details on arrangements, see [2, 9]. For each $s = 0, \ldots, k-3$, let

$$U^{(s)} = \bigcup\{P_{j_1} \cap \ldots \cap P_{j_s} \mid 1 \leq j_1 < j_2 < \ldots < j_s \leq k\}.$$

Furthermore, put $C^{(s)} = \mathbb{R}^3 \setminus U^{(s)}$ and $L^{(s-1)} = \partial C^{(s)} = \partial U^{(s)}$. We call $L^{(s)}$ the *s-th level* of the arrangement $\mathcal{A}(\mathcal{P})$. We also denote by $\bar{C}^{(s)}$ the closure of $C^{(s)}$. Note that $U^{(1)} = U$ is just the union of the polyhedra of $\mathcal{P}$, $C^{(1)} = C$ is their common exterior, and $L^{(0)}$ is the common boundary of U and C. These definitions are illustrated in the planar case in Figure 1.

In this section we prove some geometric and topological properties of $U^{(2)}$ and $U^{(3)}$, which will be used in the proof of our main theorem (Theorem 4.2 in Section 4).

Lemma 3.1 *The number of local minima and the number of connected components of $U^{(3)}$, for any collection of k polyhedra with a total of n faces in general position, are bounded by the number of local maxima of $\bar{C}^{(1)}$ plus $O(k^2)$.*

Proof: $U^{(3)}$ is the union of triple intersections of polyhedra in $\mathcal{P}$. Thus a local minimum of $U^{(3)}$ is necessarily a local minimum of one of these triple intersections. (This immediately bounds the number of local minima by $\binom{k}{3}$.) Since the number of all pairwise intersections of polyhedra in $\mathcal{P}$ is only $\binom{k}{2}$, it suffices to consider only those minima that are formed as intersections of three distinct polyhedron boundaries. By the general position assumption, such an intersection must be the unique common point v of the relative interiors of three faces of three distinct polyhedra in $\mathcal{P}$. In this case, it is easy to see that v must be a local maximum of $\bar{C}^{(1)}$. The lemma follows. We omit the rather easy details. □

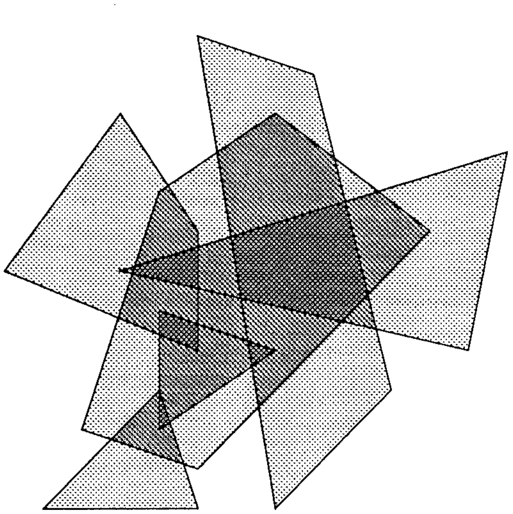


Figure 1: A family of convex polygons; $U^{(1)}$, $U^{(2)}$, and $U^{(3)}$ are shaded successively darker.

Applying Theorem 2.5, we thus obtain:

Corollary 3.2 *If $\mathcal{P}$ is a collection of polyhedra with a total of n faces, arising as Minkowski expansions of a collection of k convex polyhedra, with pairwise-disjoint interiors, by another convex polyhedron, then the number of local minima and the number of connected components of $U^{(3)}$ is $O(nk)$.*

We next study the topological structure of $U^{(2)}$. We begin by deriving a topological property of general polyhedral sets. Let S be a closed polyhedral set in general position; in particular, no two vertices of S have the same z-coordinate. Note that a neighborhood of a point p in S, as defined in the previous section, is star-shaped with respect to p, so its intersection with a horizontal plane through p is also star-shaped and thus connected.

A *merge point* of S is a point $p \in S$ such that, for any neighborhood of p, any horizontal plane lying below p and sufficiently close to it intersects the neighborhood in a disconnected set. We define the *merge number* $m(p)$ of p to be the number of connected components in this intersection, less 1. By star-shapedness of neighborhoods, $m(p)$ is well defined. (We can naturally extend this definition to any non-merge point p' by putting $m(p') = 0$.) We will apply our analysis to $S = U^{(2)}$, so in this case $m(p) \leq 2$ for any point in the set, as is easily implied by the general position assumption on $\mathcal{P}$.

The *first Betti number* of S, $\beta_1(S)$, is the rank of the first singular homology group of S [11]. Informally, it is the number of 'linearly-independent' homotopy classes of closed cycles in S, where each class consists of all cycles homotopic within S to some given cycle and not contractible in S.

Proposition 3.3 *The first Betti number $\beta_1(S)$ of any compact polyhedral set S in general position does not exceed the sum of the merge numbers of its vertices.*

Proof: Let $z : S \rightarrow \mathbb{R}$ be the z-coordinate function. For $a, b \in \mathbb{R}$, let $S[a] = z^{-1}(\{a\})$, $S[a,b] = z^{-1}([a,b])$, and $S_{\leq}[a] = z^{-1}((-\infty, a])$.

We begin the proof by stating a polyhedral analogue of a basic fact of Morse theory (cf. [18, Theorem 3.1]). The proof is an easy and standard exercise in topology, we omit it in this version.

Fact 3.4 *Let S be a triangulated polyhedral set in general position. If $S[a,b]$ contains either no vertex of S, or exactly one vertex v and the z-coordinate of v is a, then $S[a]$ is a deformation retract*[1] *of $S[a,b]$.*

We now triangulate S. This may add new vertices to S, but their presence does not affect the statement of the proposition, since they all have merge number 0. Let $z = c$ be a plane below all the vertices of S. Since S is assumed to be bounded, we have $S_{\leq}[c] = \emptyset$. It suffices to prove that, as t increases from c to $+\infty$, the Betti number $\beta_1(S_{\leq}[t])$ increases at (the z-coordinate of) each vertex of S by at most the merge number of the vertex, and never changes otherwise.

Fact 3.4 implies that $\beta_1(S_{\leq}[t])$ changes only when the plane $z = t$ sweeps through a vertex of S; indeed, if $t' > t$ is such that no vertex of S has z coordinate in $(t, t']$, then $S_{\leq}[t]$ is a deformation retract of $S_{\leq}[t']$, and thus $S_{\leq}[t]$ and $S_{\leq}[t']$ have the same homologies [21, Corollary 1.12]. Let v be a vertex of S. We want to bound the difference $\beta_1(S_{\leq}[t]) - \beta_1(S_{\leq}[t'])$, where $t = z(v)$ and $t' < t$ is appropriately close to t.

One can easily verify that the transformation from $S_{\leq}[t']$ to $S_{\leq}[t]$ is equivalent, topologically, to gluing a ball D centered at v to $S_{\leq}[t']$, choosing the size of D to be such that it intersects all triangles of $T_{t'}$ at least one of whose vertices approaches v as $t' \rightarrow t$, and no other triangles. An easy topological calculation[2] yields

$$\beta_1(S_{\leq}[t]) = \beta_1(S_{\leq}[t'] \cup D) \leq$$

$$\beta_1(S_{\leq}[t']) + \beta_1(D) + \beta_0(S_{\leq}[t'] \cap D) - 1.$$

Here $\beta_0(X)$ is the number of connected components of X. Now $\beta_1(D) = 0$ and $\beta_0(S_{\leq}[t'] \cap D)$ is exactly equal to the number of components of the intersection of a neighborhood of v in S with the plane $z = t'$, lying just below v, namely, it is equal to $m(v) + 1$. Hence $\beta_1(S_{\leq}[t]) - \beta_1(S_{\leq}[t']) \leq m(v)$, as claimed. This concludes the proof of Proposition 3.3. □

Lemma 3.5 *For a collection of k arbitrary convex polyhedra in general position, with a total of n faces, $\beta_1(U^{(2)})$ is at most proportional to k^2 plus the number of local minima of $U^{(3)}$. Hence, for collections of Minkowski expansions, as above, this number is $O(nk)$.*

Proof: Applying Proposition 3.3 to $S = U^{(2)}$, we conclude that it suffices to bound the number of merge points of $U^{(2)}$. This is because the merge number of any point of $U^{(2)}$ is at most 2, as follows from the general position assumption (and already noted above). Let $v \in \partial U^{(2)}$ be a merge point; then v must be a vertex of $\mathcal{A}(\mathcal{P})$. We may assume that v is a vertex formed by the transversal intersection of some three faces, F_i, F_j, F_ℓ, belonging to three respective distinct expanded polyhedra P_i, P_j, P_ℓ. (Indeed, any other vertex is either a vertex of some P_i or of some $P_i \cap P_j$, for $i, j = 1, \ldots, k$. No vertex of the former type can be a merge point of any $U^{(s)}$, as is easily seen. A vertex of the latter type, on the other hand, can be a merge point only if it is the bottommost point of $P_i \cap P_j$, as is readily verified using the general position assumption. Thus the number of these merge points is at most $\binom{k}{2}$.) An easy case analysis shows that any vertex v of this form can be a merge point of $U^{(2)}$ only if it is a local minimum of $U^{(3)}$. We omit the details. □

4 The Complexity of the Union

Having all this machinery available, we now turn to the analysis of the complexity of U. The initial portion of the analysis is identical to that of [4], and here we only go quickly through its main steps; we refer the reader to that paper for full details. Specifically, let $\psi(k, n)$ denote the total number of edges of U contained in $\bigcup_{1 \leq i < j \leq k} \partial P_i \cap \partial P_j$, maximized over all collections $\mathcal{P}$ of k Minkowski sums, as above, with a total of n faces, in general position. We apply the inductive analysis scheme of [4], which yields the following recurrence for $\psi(k, n)$:

$$(k-2)\psi(k,n) \leq \sum_{i=1}^{k} \psi(k-1, n-n_i) + \frac{1}{2}\psi^{(1)}(k,n) \ , \quad (1)$$

where n_i is the number of faces of P_i, for $i = 1, \ldots, k$, and where $\psi^{(1)}(k, n)$ is the maximum, taken over all collections $\mathcal{P}$ as above, of the quantity $\sum_f w(f)$, with the sum ranging over all faces f of $L^{(1)}$; here $w(f) = \max(4e_f - 6, 0)$ and e_f is the number of edges on ∂f that lie on the boundary of the only polyhedron containing f

[1] For a topological space X, a subspace $Y \subseteq X$ is called a *deformation retract* of X if there exists a continuous map $\rho : X \times [0,1] \rightarrow X$ such that $\rho(x, 0) = x$, for all $x \in X$, $\rho(X, 1) = Y$, and $\rho(y, 1) = y$ for all $y \in Y$; see, for example, [21, pp. 15,19].

[2] It is easy to show that, for two path-connected topological spaces X and Y, $\beta_1(X \cup Y) \leq \beta_1(X) + \beta_1(Y) + \beta_0(X \cap Y) - 1$. An elementary proof of this fact can be found in [15]. If X is not path-connected, but Y is, the statement follows by repeated applications of the path-connected version. Note that in our case D is path-connected but $S_{\leq}[t']$ need not be. The statement also easily follows from arguments based on the Mayer-Vietoris sequence; see, for example, Vick [21, p. 22].

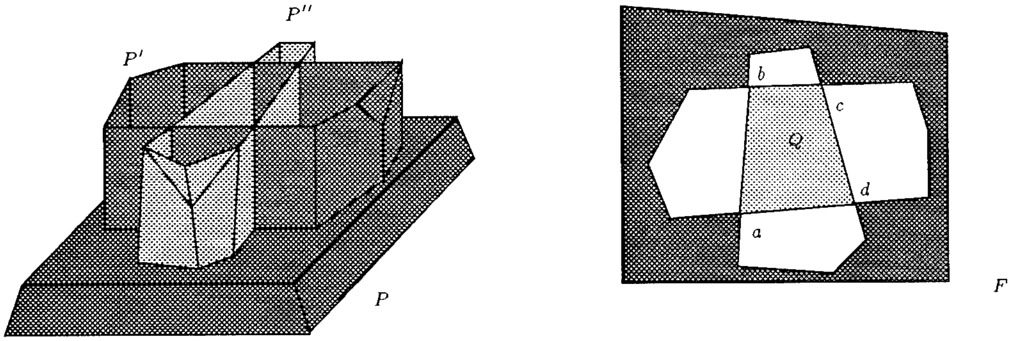


Figure 2: A special quadrilateral Q.

in its interior. (This definition of $\psi^{(1)}(k,n)$ is not used here, and is given only for the sake of completeness.)

We next estimate $\psi^{(1)}(k,n)$, exactly as in [4], by using the same inductive scheme. This leads to the recurrence

$$(k-2)\psi^{(1)}(k,n) = \sum_{i=1}^{k} \psi^{(1)}(k-1, n-n_i) + O(nk+\psi^{(2)}(k,n)) \ , \tag{2}$$

where $\psi^{(2)}(k,n)$ is the number of "special quadrilaterals" of the following form, maximized over all collections $\mathcal{P}$ as above. Such a quadrilateral Q is schematically depicted in Figure 2; it is defined by a triple $(F(Q), P'(Q), P''(Q))$, where $P' = P'(Q)$, $P'' = P''(Q)$ are distinct polyhedra of $\mathcal{P}$, and $F = F(Q)$ is a face of another polyhedron, $P(Q)$, of $\mathcal{P}$; the intersection $P' \cap P'' \cap F$ is the quadrilateral Q, which is assumed to be disjoint from all other polyhedra, the four vertices of Q are vertices of the union U, two opposite edges of Q are contained in $\partial P' \cap F$ and in the interior of only P'', and the two other opposite edges of Q are contained in $\partial P'' \cap F$ and in the interior of only P'. In other words, if we remove P' and P'' from $\mathcal{P}$ and replace them by $P' \cap P''$, then ∂Q appears on the boundary of the union of the modified collection, and the union of all polyhedra in $\mathcal{P}$, except for P' and P'', contains Q on its boundary. In the notation of Section 3, Q is a special type of face of $L^{(2)}$. The structure of $U^{(2)}$ locally near Q is schematically depicted in Figure 3.

In the case of general convex polyhedra, the authors give a bound of $O(k^3 + nk)$ for $\psi^{(2)}(k,n)$, and show that it is tight in the worst case [4]. However, we show here that the special properties of $\mathcal{A}(\mathcal{P})$ in the case of Minkowski sums, as established in Section 3, lead to an improved bound of only $O(nk)$. Plugging this bound into the recurrences (1) and (2), we obtain, following the analysis of [4], $\psi^{(1)}(k,n) = O(nk \log k)$ and $\psi(k,n) = O(nk \log^2 k)$, which thus completes the proof of the upper bound of our main result.

We actually prove the following stronger result:

Lemma 4.1 *For collections $\mathcal{P}$ of arbitrary convex polyhedra in general position, the number of special quadrilaterals is at most proportional to nk plus the number of local minima of $U^{(3)}$. Hence, for Minkowski expansions, as above, this number is $O(nk)$.*

Proof: Let Q be a special quadrilateral defined by $(F(Q), P'(Q), P''(Q))$, as above. The boundary ∂Q of Q is necessarily a bounding cycle of a connected component of $\partial(P' \cap P'') \cap L^{(1)}$. Following an argument similar to that of [4], we observe that the overall number of special quadrilaterals Q whose boundaries are contractible to a point in the corresponding sets $\partial(P'(Q) \cap P''(Q)) \cap L^{(1)}$ is just $O(nk)$, since in the process of contracting ∂Q one has to encounter a vertex of $P'(Q) \cap P''(Q)$, and it is easily checked that no vertex of this form is charged by more than one contractible quadrilateral, for fixed P' and P''. The claim now follows from Proposition 2.1. We thus need to consider only *non-trivial* quadrilaterals, namely those whose boundary is not contractible in the above fashion.

Let Q be such a non-trivial special quadrilateral. By definition, Q does not meet the interior of $U^{(3)}$, but is

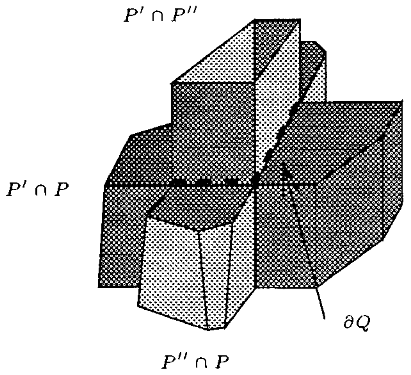


Figure 3: $U^{(2)}$ near a special quadrilateral Q.

fully contained in $U^{(2)}$. Moreover, $\partial Q \subset \partial U^{(2)}$ (see Figure 3). "Cut" $U^{(2)}$ along Q (or, rather, shift Q slightly away from $P(Q)$ and then cut $U^{(2)}$ along the shifted quadrilateral). We apply this cutting procedure to every non-trivial special quadrilateral, in some arbitrary order. Since, by definition, the relative interiors of any pair of distinct special quadrilaterals are disjoint, it follows that the cuts do not interfere with each other.

Each cut of $U^{(2)}$, performed along some non-trivial special quadrilateral Q, has one of two possible effects: Either the first Betti number of $U^{(2)}$ decreases by 1, or the number of connected components of $U^{(2)}$ increases by 1, according to, respectively, whether points of $U^{(2)}$ lying on the two sides of Q sufficiently near Q can or cannot be connected by a path that avoids Q in the current version of $U^{(2)}$. The number of cuts is thus proportional to the increase in the number of connected components of $U^{(2)}$ plus the decrease in the first Betti number of $U^{(2)}$, as effected by the cuts. Since none of the cuts can *increase* the first Betti number of $U^{(2)}$, it follows, by Lemma 3.5, that the latter quantity (the decrease in $\beta_1(U^{(2)})$) is bounded by $O(k^2)$ plus the number of local minima of $U^{(3)}$.

To estimate the former quantity, consider a cut performed along a special quadrilateral Q, defined by the triple $(F(Q), P'(Q), P''(Q))$, which increases the number of components of the current version of $U^{(2)}$. Since Q is non-trivial, "dragging" Q along $\partial(P'(Q) \cap P''(Q))$ in either direction encounters a third polyhedron, and thus also a distinct component of $U^{(3)}$. To be more precise, one of the new components of $U^{(2)}$, denoted $U_1^{(2)}$, contains, near Q, points belonging to $P(Q) \cap P'(Q) \cap P''(Q)$, so $U_1^{(2)}$ contains a component of $U^{(3)}$. The other component of $U^{(2)}$, denoted $U_2^{(2)}$, is bounded, near Q, by the connected portion K of $\partial(P'(Q) \cap P''(Q)) \cap L^{(1)}$ incident to Q. It is clear that K must also be incident (before any cut was made) to some third polyhedron, for otherwise K would have been homeomorphic to a disk whose boundary corresponds to ∂Q, and Q would then have been a trivial quadrilateral, contractible to a point in $K \subset \partial(P'(Q) \cap P''(Q)) \cap L^{(1)}$. There are now two subcases to consider:

(i) The component $U_2^{(2)}$ also contains a component of $U^{(3)}$. (Note that no component of $U^{(3)}$ is ever split by the cuts, because all special quadrilaterals are disjoint from the interior of $U^{(3)}$, and the cuts, performed along slightly shifted copies of the quadrilaterals, are thus disjoint from $U^{(3)}$.) The total increase in the number of components of $U^{(2)}$ formed by cuts of this kind cannot exceed the number of components of $U^{(3)}$, and can thus be bounded by the number of local minima of $U^{(3)}$.

(ii) The component $U_2^{(2)}$ does not contain a component of $U^{(3)}$. Since the corresponding boundary K, as defined above, was incident, before any cut was made, to another component of $U^{(3)}$, it follows that this subcase occurs because previous cuts, along other special quadrilaterals incident to K, have separated K from all other adjacent components of $U^{(3)}$. In this case, we charge the current cut to one of these preceding cuts. It is easily checked that no cut is charged in this manner more than once. This implies that the number of special quadrilaterals in this subcase is at most equal to the number of non-trivial special quadrilaterals of the preceding types (those decreasing $\beta_1(U^{(2)})$ and those appearing in case (i) above).

This completes the proof for collections of general convex polyhedra. In the case of Minkowski expansions, as above, Corollary 3.2 implies that the number of special quadrilaterals is $O(nk)$. □

We have thus shown that, in the case of Minkowski expansions, $\psi^{(2)}(k, n) = O(nk)$, which in turn completes the analysis of $\psi(k, n)$ and yields the upper bound in the main result of the paper:

Theorem 4.2 *Let $A_1, \ldots, A_k$ be k convex polyhedra in 3-space with pairwise disjoint interiors, and let B be another convex polyhedron. The combinatorial complexity of the union of the Minkowski sums $A_i \oplus (-B)$, for $i = 1, \ldots, k$, is $O(nk \log^2 k)$, where n is the overall complexity of the individual Minkowski sums. In the worst case, this complexity can be $\Omega(nk\alpha(k))$.*

Remark: The preceding argument assumes general position of the polyhedra A_i and B. Nevertheless, the theorem also holds for collections of polyhedra not in general position, as can be argued using a perturbation scheme [4]. We omit the details in this version.

Proof of the lower bound: We make use of a planar construction, given in [1], of k convex polygons with a total of n edges, so that their union has $\Omega(n\alpha(k))$ edges and vertices. Additionally, the polygons can be arranged so that their union is star-shaped, say with respect to the origin, and at least some fixed fraction of its vertices are visible from the point $(0, +\infty)$, in the strong sense that there exists some fixed angle $\beta > 0$ (independent of k and n), so that, for any such visible vertex v, the wedge whose apex is v, whose bisecting ray is parallel to the positive y-axis, and whose angle is β, does not meet any of the polygon interiors.

Without loss of generality, assume that k is even and $n \geq 4.5k$. We start our three-dimensional construction with a set of $k/2$ convex polygons in xy-plane with $n - 3k$ edges altogether, so that their union U' has $\Omega(n\alpha(k))$ vertices visible from $(0, +\infty, 0)$ in the above strong sense. By scaling, we may assume that the entire construction is contained in the unit disk about the origin in the xy-plane. Now slightly shift and expand the polygons in the z-direction, each by a different amount, to produce pairwise-disjoint flat and thin convex prisms, all contained in the slab $|z| \leq 0.1$, say; see Figure 4. The second set of $k/2$ polyhedra consists of points $(0, M, i)$,

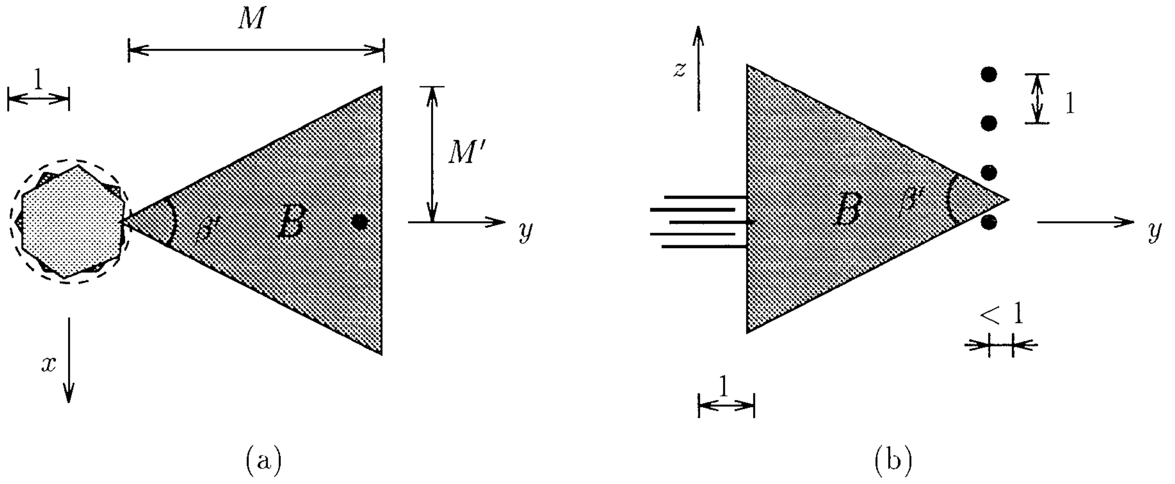


Figure 4: The lower bound construction, not to scale: (a) a view from above; (b) a side view.

for $i = 1, \ldots, k/2$ (or, rather, tiny tetrahedra centered around these points), where $M \gg k$ is an appropriate parameter. This gives us a collection $\{A_i\}_{i=1}^{k}$ of k pairwise-disjoint convex polyhedra with a total of n faces. The polyhedron B is a tetrahedron with vertices $(0, 0, \pm M')$ and $(\pm M', M, 0)$, where $M' < M/4$ is another parameter, chosen so that the dihedral angles of B at its horizontal and vertical edges are both equal to some $\beta' < \beta$; note that, by the choice of M', we have $\tan \frac{\beta'}{2} < 1/4$, an inequality that will be needed below. See Figure 4 for an illustration.

Let v be a vertex of U' visible from $(0, +\infty, 0)$ in the above sense. By construction, we can place B so that its vertical edge e_v touches the two prisms corresponding to the two polygons whose boundaries intersect at v; moreover, we can slide B vertically upwards and downwards, by a total distance of close to $2M'$, so that e_v maintains these two contacts, while the interior of B remains disjoint from any of the shifted prisms. It is easily seen that, for an appropriate choice of M, independently of the choice of v, the boundary of B will meet each of the tiny tetrahedra A_i around the points $(0, M, i)$ during the vertical motion. Moreover, our choice of parameters also implies that the intersection of B with the vertical line $x = 0, y = M$ has length less than $2 \tan \frac{\beta}{2} < 1/2$, so, when B touches one of these A_i's, its interior remains disjoint from all the other polyhedra A_i; see Figure 4(b) for an illustration. In other words, each of the appropriate $\Omega(n\alpha(k))$ vertices of U' gives rise to $\Omega(k)$ placements of B where it makes three contacts with the A_i's, while its interior remains disjoint from all these polyhedra. Since each of the resulting $\Omega(nk\alpha(k))$ placements of B corresponds to a vertex of the union of the expanded polyhedra $\bigcup_{i=1}^{k} A_i \oplus (-B)$, the lower bound of the theorem follows. □

Corollary 4.3 *The combinatorial complexity of the free configuration space of a convex polyhedron B, translating in 3-space amidst a collection of k convex obstacles, $A_1, \ldots, A_k$, having pairwise disjoint interiors, is $O(nk \log^2 k)$ and can be $\Omega(nk\alpha(k))$ in the worst case, where n is the overall complexity of the Minkowski sums $A_i \oplus (-B)$, for $i = 1, \ldots, k$.*

Remarks: (1) Going back to the parameters p and q that count the number of faces of B and of all the A_i's together, we can state the upper bound of Theorem 4.2 and Corollary 4.3 as $O(pqk \log^2 k)$, and the lower bound as $\Omega(qk\alpha(k))$ for constant p. A sharper dependence of the bound on p (when q and k are fixed) remains an open problem.

(2) Our proof of Theorem 4.2 makes use of the fact that the P_i's are Minkowski sums only in the proof of Theorem 2.4. Hence our analysis also applies to any collection of k general convex polyhedra with a total of n faces, with the property that, for any subset of r of these polyhedra, the number of components of the complement of their union is $O(rm)$, where m is the total number of faces of those r polyhedra. We pose it as an open problem to find other natural examples of collections of convex polyhedra with this property.

5 Efficient Construction of the Union

We next describe an efficient randomized algorithm for constructing the union U of a collection $\mathcal{P}$ of expanded polyhedra, as above. The input to the algorithm consists of the original polyhedra A_i and B, so we first compute the individual Minkowski sums $P_i = A_i \oplus (-B)$, for $i = 1, \ldots, k$. This computation can be done in several ways, the most efficient of which is by using the technique of Guibas and Seidel [13]. Each P_i can be constructed in time $O(p+q_i+n_i)$, where n_i is the complexity of P_i. Thus the cost of this stage is $O(pk+q+n)$.

The main algorithm is essentially identical to the one given in [4] for the case of general polyhedra; for the sake of completeness, we briefly review the algorithm here, and revise its analysis to exploit the improved combinatorial bounds derived above.

The algorithm proceeds as follows. We first compute all the pairwise intersections $P_i \cap P_j$, for $1 \le i < j \le k$. This can be done in time $O(nk)$, as shown in [4]. In additional $O(nk)$ time, we can also extract, for each face F of a polyhedron $P_i \in \mathcal{P}$, the collection $\mathcal{Q}_F$ of the convex polygons $Q_j = F \cap P_j$, for $j \neq i$. The set $U_F = F \setminus \bigcup_{j\neq i} Q_j$ is the portion of F that appears on ∂U, so the algorithm computes the sets U_F, over all faces F. Reconstructing the boundary of U from this information is relatively straightforward, by gluing the sets U_F to each other in an appropriate manner. The construction of the sets U_F is done exactly as in [4]: For each face F of some $P_i \in \mathcal{P}$, we choose a random order of the polyhedra in $\mathcal{P} \setminus \{P_i\}$ and insert the polygons $Q_j \in \mathcal{Q}_F$, one by one, in the corresponding order, maintaining the complement of their union (within the plane containing F) as we go. For this we use the same technique as in [7, 12, 17], which maintains a vertical decomposition (relative to some direction within F) of the complement into trapezoids; see [4] for more details. At the end of the incremental procedure, it is relatively straightforward to truncate the resulting complement to within F. The cost of this truncation is easily seen to be proportional to the complexity of F plus the number of final trapezoids forming the complement of $\bigcup \mathcal{Q}_F$. Thus this step does not increase the asymptotic complexity of the algorithm. In the motion planning application, mentioned below, we also need support for efficient point location queries in U_F. No additional processing is needed, as the data structure used to compute U_F, as adapted from the algorithms of [7, 17], already supports logarithmic-time point location queries.

The cost of the algorithm is proportional to the number of trapezoids that are created during the execution of the algorithm, plus the sum of their weights, plus the initial cost $O(nk)$ of computing all the collections $\mathcal{Q}_F$. Here the *weight* of a trapezoid $\tau \subseteq F$ is the number of polygons $Q_j \in \mathcal{Q}_F$ that intersect the relative interior of τ; each of them either fully contains τ or its boundary crosses τ. Define a *canonical trapezoid* $\tau \subseteq F$ to be a trapezoid that occurs in the trapezoidal decomposition of the complement (within the plane containing F) of the union of some subset of $\mathcal{Q}_F$. Such a trapezoid is defined by between 1 and 5 polyhedra—P_i plus the at most 4 polyhedra that define its sides and corners in the plane containing F. The following argument applies only to canonical trapezoids defined by exactly 5 polyhedra. It has to be repeated, with straightforward modifications, for trapezoids defined by 1, 2, 3, or 4 polyhedra. Adapting the analysis of [4], and using the Clarkson-Shor probabilistic technique [8], one easily shows:

Claim 1: The probability that a canonical trapezoid τ, lying in the plane containing F and having weight w, is created during the incremental construction is $1/\binom{w+4}{4}$.

Claim 2: The overall number T_w of canonical trapezoids with weight less than w, over all faces F of the given polyhedra, is $O(w^3 nk \log^2 \frac{k}{w})$.

The expected running time of the algorithm, over all faces F, is thus

$$O\left(nk + \sum_{w=0}^{k-5} \frac{(w+1)t_w}{\binom{w+4}{4}}\right) ,$$

where t_w is the total number of canonical trapezoids whose weight is exactly w. Since $t_w = T_{w+1} - T_w$ (with $T_0 = 0$), we can rewrite the above sum as

$$O\left(\cdot nk + \sum_{w=0}^{k-5} \frac{T_{w+1} - T_w}{(w+4)(w+3)(w+2)}\right) =$$

$$O\left(nk + \sum_{w=1}^{k-5} \frac{w^3 nk \log^2 \frac{k}{w}}{(w+4)(w+3)(w+2)(w+1)}\right) =$$

$$O\left(nk + \sum_{w=1}^{k-5} \frac{nk}{w} \log^2 \frac{k}{w}\right) = O(nk \log^3 k) .$$

Hence we have:

Theorem 5.1 *The union of a collection of k expanded convex polyhedra in 3-space, as above, with a total of n faces, can be computed in randomized expected time $O(nk \log^3 k)$.*

We omit here the rather easy details of the following application to motion planning:

Corollary 5.2 *Given a convex polyhedron B, free to translate among k convex polyhedral obstacles with pairwise-disjoint interiors, the entire free configuration space of B can be computed and preprocessed in randomized expected time $O(nk \log^3 k)$, where n is the total number of vertices of the Minkowski sums of the obstacles and of $-B$. Then, given two placements, ζ_1, ζ_2, of B, we can decide, in $O(k \log n)$ time, whether B can translate in a collision-free manner from ζ_1 to ζ_2.*

6 Conclusions

In this paper we have shown that the combinatorial complexity of the union of the Minkowski sums of k convex polyhedra in three dimensions, having pairwise-disjoint interiors, with another convex polyhedron, is $O(nk \log^2 k)$ (and $\Omega(nk\alpha(k))$ in the worst case), where

n is the overall complexity of the individual Minkowski sums. We have also presented an efficient and rather simple randomized algorithm for computing the union in expected time $O(nk \log^3 k)$. Both the combinatorial bound and the algorithm have applications to translational motion planning of a convex polyhedral object in a 3-dimensional environment amidst polyhedral obstacles, and we have also discussed these applications.

These results almost settle a long-standing open problem, but raise a whole collection of new open problems (some of these problems have already been mentioned in earlier sections). One open problem is to tighten the remaining gap between the lower and upper bounds on the complexity of the union. We conjecture that the correct worst-case bound is $\Theta(nk\alpha(k))$. There are also the problems of designing an efficient deterministic algorithm for computing the union, and of improving the query performance of the motion planning algorithm mentioned above.

The more challenging and interesting open problems, however, involve generalizations and extensions of our results and techniques. First, what is the combinatorial complexity of the union of Minkowski sums $A_i \oplus B$, where the A_i's are k convex polyhedra with pairwise disjoint interiors, and B is a ball? Even the special case where the A_i's are lines seems to be open; in this case we want to bound the combinatorial complexity of the union of k congruent infinite cylinders, where the conjecture is that this complexity is near-quadratic in k. This problem arises in motion planning, when applying a standard heuristic of enclosing the moving (rigid) object by a ball, and planning the motion of the enclosing ball.

Acknowledgements

We wish to thank Dan Halperin for useful discussions on the problem, and Jiří Matoušek and Igor Rivin for helpful suggestions on matters of topology.

References

[1] B. Aronov and M. Sharir, The common exterior of convex polygons in the plane, manuscript, 1994.

[2] B. Aronov and M. Sharir, Triangles in space, or building (and analyzing) castles in the air, *Combinatorica* 10(2) (1990), 137–173.

[3] B. Aronov and M. Sharir, Castles in the air revisited, *Proc. 8th ACM Symp. on Computational Geometry*, 1992, pp. 146–156. Also to appear in *Discrete Comput. Geom.*

[4] B. Aronov and M. Sharir, The union of convex polyhedra in three dimensions, *Proc. 34th IEEE Symp. on Foundation of Computer Science*, 1993, pp. 518–529.

[5] M. de Berg, J. Matoušek, and O. Schwarzkopf, Piecewise linear paths among convex obstacles, *Proc. 25th ACM Symp. on Theory of Computing,* 1993, pp. 505–514.

[6] B. Chazelle, An optimal algorithm for intersecting three-dimensional convex polyhedra, *SIAM J. Computing* 21(4)(1992), 671–696.

[7] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir, and J. Snoeyink, Computing a single face in an arrangement of line segments and related problems, *SIAM J. Computing* 22 (1993), 1286–1302.

[8] K. Clarkson and P. Shor, Applications of random sampling in computational geometry II, *Discrete Comput. Geom.* 4 (1989), 387–421.

[9] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[10] H. Edelsbrunner, L. Guibas, and M. Sharir, The upper envelope of piecewise linear functions: Algorithms and applications, *Discrete Comput. Geom.* 4 (1989), 311–336.

[11] M. Greenberg and J. Harper, *Algebraic Topology: A First Course*, Benjamin-Cummings, Reading, MA, 1981.

[12] L. Guibas, D. Knuth, and M. Sharir, Randomized incremental construction of Voronoi and Delaunay diagrams, *Algorithmica* 7 (1992), 381–413.

[13] L. Guibas and R. Seidel, Computing convolutions by reciprocal search, *Discrete Comput. Geom.* 2 (1987), 175–193.

[14] D. Halperin and C.K. Yap, Combinatorial complexity of translating a box in polyhedral 3-space, *Proc. 9th ACM Symp. on Computational Geometry,* 1993, 29–37.

[15] G. Hotz and J. Sellen, On algebraic computation trees and Betti numbers, Manuscript, 1993.

[16] K. Kedem, R. Livne, J. Pach and M. Sharir, On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles, *Discrete Comput. Geom.* 1 (1986), 59–71.

[17] N. Miller and M. Sharir, Efficient randomized algorithms for constructing the union of fat triangles and of pseudo-disks, manuscript, 1991.

[18] J. Milnor, *Morse Theory,* Princeton University Press, Princeton, NJ, 1963.

[19] J. Pach and M. Sharir, The upper envelope of piecewise linear functions and the boundary of a region enclosed by convex plates: Combinatorial analysis, *Discrete Comput. Geom.* 4 (1989), 291–309.

[20] M. Sharir, Efficient algorithms for planning purely translational collision-free motion in two and three dimensions, *Proc. IEEE Symp. on Robotics and Automation,* 1987, pp. 1326–1331.

[21] J.W. Vick, *Homology Theory: An Introduction to Algebraic Topology,* Academic Press, New York, 1973.

Motion Planning amidst Fat Obstacles *

(extended abstract)

A. Frank van der Stappen † Mark H. Overmars †

Abstract

We present an efficient and simple paradigm for motion planning amidst fat obstacles. The paradigm fits in the cell decomposition approach to motion planning and exploits workspace properties that follow from the fatness of the obstacles. These properties allow us to decompose the workspace, subject to some constraints, rather than to decompose the higher-dimensional free space directly. A sequence of uniform steps transforms the workspace decomposition into a free space decomposition of asymptotically the same (expectedly small) size. The approach applies to robots with any fixed number of degrees of freedom and turns out to be successful in many cases: it leads to nearly optimal $O(n \log n)$ algorithms for motion planning in 2D, and for motion planning in 3D amidst obstacles of comparable size. In addition, we obtain algorithms for planning 3D motions among polyhedral obstacles, running in $O(n^2 \log n)$ time, and among arbitrary obstacles, running in time $O(n^3)$.

1 Introduction

Fatness turns out to be an interesting phenomenon in computational geometry. Several papers discuss the surprising impact of fatness of the objects under consideration on combinatorial complexities and algorithm efficiencies. Fat objects are compact to some extent, rather than long and thin. Examples of the influence of fatness are described in papers by Alt et al. [1] and Matoušek et al. [5] which show that the combinatorial complexity of the union of geometric figures is low if the figures are fat. Overmars [6] presents an efficient algorithm for point location in fat subdivisions.

Robot motion planning considers the problem of finding a collision-free path for a robot $\mathcal{B}$ in a workspace W with obstacles $\mathcal{E}$ from an initial to a final placement, i.e., a path that avoids collision of $\mathcal{B}$ with the obstacles. The motion planning problem is often studied as a problem in the configuration space C, which is the space of parametric representations of robot placements. The free space FP is the subspace of C of points that represent placements of the robot in which it does not intersect any obstacle in $\mathcal{E}$. The problem of finding a collision-free path for $\mathcal{B}$ corresponds to finding a continuous curve in FP (or its closure) connecting the points representing the initial and final robot placement. The combinatorial complexity of FP largely influences the complexity of finding such a curve.

In [11], we examine the role of fatness in motion planning. We show that the combinatorial complexity of the free space FP is linear in the number of obstacles if these (stationary) obstacles are fat and satisfy certain weak additional assumptions, whereas it would be much higher if the obstacles are not fat. The additional assumptions mainly require the obstacles and the robot to be of constant complexity, and the robot to be not too large compared to the obstacles. The obstacle fatness and the additional assumptions provide a realistic framework for efficient motion planning.

A question that immediately comes to mind when considering the combinatorial result of [11] is whether this reduced complexity opens the way to efficient motion planning algorithms for such realistic environments. To answer this important question, we have considered various existing (planar) motion planning algorithms and examined the influence of obstacle fatness on their efficiency. The boundary-vertices retraction algorithm by Sifrony and Sharir [10] for a ladder moving in a planar workspace with polygonal obstacles runs in time $O(K \log n)$, where K is the number of pairs of obstacle corners that lie less than the length of the ladder apart. Fatness of the obstacles causes K to be only $O(n)$, whereas it would be $O(n^2)$ for arbitrary obstacles. The boundary cell decomposition algorithm by Avnaim, Boissonnat, and Faverjon [2], running in time $O(n^3 \log n)$ for a constant complexity polygonal robot amidst arbitrary polygonal obstacles, can be shown to run in $O(n \log n)$ time in the realistic setting of fat obstacles, due to the bounded local complexity (see e.g. [9, 10]) of the workspace caused by the fatness. The $O(n^5)$

*Research is supported by the Dutch Organization for Scientific Research (N.W.O.) and by the ESPRIT III BRA Project 6546 (PROMotion).

†Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

algorithm by Schwartz and Sharir [7] for planning the motion of a ladder or a polygonal robot amidst polygonal obstacles can be shown to run, unmodified, in time $O(n^2)$ if the obstacles are fat, whereas a minor modification enhances the efficiency to a running time of $O(n \log n)$ [12, 13].

Algorithms for efficient motion planning in 3D workspaces are scarce: approaches in contact space, like the algorithms mentioned above by Sifrony and Sharir, and by Avnaim, Boissonnat, and Faverjon, were never shown to generalize to higher dimensions. General approaches to motion planning (e.g. by Schwartz and Sharir [8]) are computationally expensive. Three-dimensional workspaces imply at least three-dimensional configuration spaces with arrangements defining the free portions. Naturally, the structure of such higher-dimensional arrangements is considerably more complex to understand, let alone to subdivide the free arrangement cells into simple subcells or catch their structure in some one-dimensional roadmap. At this point, however, fatness comes to our help to provide us with a very beneficial property of the workspace, which in fact also led to the enhanced performance of the planar motion planning algorithms: the bounded local complexity of the workspace implied by the fatness of the objects makes it possible to partition the workspace W (rather than the configuration space) into regions R such that the free part of the configuration space cylinder obtained by lifting R into configuration space has constant complexity. Moreover, the bounded local complexity also establishes existence of small partitions into such regions.

We formalize and exploit the workspace properties outlined above to obtain a paradigm for planning the motion of a constant complexity robot moving amidst constant complexity fat obstacles. The paradigm follows the cell decomposition approach to motion planning. The size of the resulting decomposition of the free space into simple subcells is determined by the size of some constrained workspace partition. The running time of algorithms based on the paradigm depends on the time to find such a partition. In Section 2 we discuss some geometric preliminaries. Section 3 explains the paradigm and its context and shows how its performance depends on the size of a constrained partition of the workspace. Section 4 discusses the construction of small partitions for 2D workspaces with arbitrary fat obstacles and 3D workspaces with polyhedral fat obstacles, leading to $O(n \log n)$ and $O(n^2 \log n)$ motion planning algorithms respectively, independent of the number of degrees of freedom of the robot. We end the section by mentioning a nice additional result, and, finally, draw some conclusions in Section 5.

2 Preliminaries: fatness and wrappings

In this section we recall the definition of fatness introduced in [11] and formulate a lemma stating a useful object density property for a collection of fat objects placed in $\mathbb{R}^d$. Moreover, we prove a low complexity bound for arrangements of sufficiently tight wrappings of fat objects. Such arrangements play an important role throughout this paper.

2.1 Fatness

Contrary to many other definitions of fatness in literature, the definition given below applies to general shapes in arbitrary dimension. It involves a parameter k which supplies a qualitative measure of fatness: the smaller the value of k, the fatter the object must be.

Definition 2.1 [k-fatness]
Let $E \subseteq \mathbb{R}^d$ be an object and let k be a positive constant. The object E is k-fat iff for all hyperspheres S centered inside E and not fully containing E:

$$k \cdot volume(E \cap S) \geq volume(S).$$

The definition forbids fat obstacles to be long and thin or to have long or thin parts.

The following "proximity lemma" will be crucial in the rest of this paper. For a sketch of the proof we refer to [11], where a similar lemma is proven.

Lemma 2.2 *Let $\mathcal{E}$ be a set of non-intersecting k-fat objects in $\mathbb{R}^d$ and let $c \geq 0$ be a constant. Then the number of objects $E \in \mathcal{E}$ with diameter at least δ intersecting any hypercubic box with side length $c \cdot \delta$ is bounded by a constant depending on k and c.*

In other words, when all objects have diameter at least δ, any hypercube with side length $c \cdot \delta$ contains only a constant number of objects. This workspace property resembles a property that is referred to as *bounded local complexity* [9]. The motion planning paradigm outlined in the section will actually work for any set of obstacles satisfying Lemma 2.2, not only for fat obstacles.

2.2 Wrappings of fat objects

Sufficiently tight wrappings of objects play a crucial role throughout this paper. Apart from providing the justification that the paradigm for motion planning amidst fat obstacles presented here indeed works, the wrappings also help in finding efficient instances of the paradigm.

An ϵ-wrapping of an object E is an enclosing shape of E, with the property that the distance from the wrapping to E never exceeds ϵ.

Definition 2.3 [ϵ-wrapping]
Let $E \subseteq \mathbb{R}^d$ and let $\epsilon \in \mathbb{R}^+$. Any Δ satisfying $E \subseteq \Delta \subseteq \{ p \in \mathbb{R}^d \mid d(p, E) \leq \epsilon \}$ is an ϵ-wrapping of E.

The complexity of a scene of n disjoint k-fat objects is clearly $O(n)$. If we expand the objects in some way then they will start intersecting, and eventually the combinatorial complexity of the scene will increase. We may expect, however, that the complexity will remain $O(n)$ if we expand the k-fat objects by some bounded amount. Theorem 2.4 states the circumstances that lead to this result. An obvious way to express a bound on the expansion of an object E is to state that the expanded object is an ϵ-wrapping of the object E itself, for some bounded positive ϵ.

Theorem 2.4 *Let $\mathcal{E}$ be a set of n non-intersecting k-fat objects in $\mathbf{R}^d$ of constant complexity each. Let δ be a lower bound on the diameters of the objects $E \in \mathcal{E}$ and let $c \geq 0$ be some constant. Assume that a constant complexity $(c \cdot \delta)$-wrapping $\Delta(E)$ is given for every object $E \in \mathcal{E}$. Then:*

(a) the complexity of the arrangement $\mathcal{A}(\Delta)$ of all wrapping boundaries $\partial\Delta(E)$ is $O(n)$,

(b) every point $p \in \mathbf{R}^d$ lies inside at most $O(1)$ wrappings $\Delta(E)$.

Proof: We assume, to prove the (a)-part, that the objects in $\mathcal{E}$ are ordered by increasing size: $E_1, \ldots, E_n$. We intend to count for each object E_i the subspaces of dimensions 0 to $d-1$ that are defined by the intersection of $\partial\Delta(E_i)$ and wrapping boundaries $\partial\Delta(E_j)$ with $j > i$. Two $(c \cdot \delta)$-wrapping boundaries $\partial\Delta(E_i)$ and $\partial\Delta(E_j)$ $(i < j)$ can only intersect if the objects E_i and E_j are less than $2c \cdot \delta$ apart. By the fact that δ is a lower bound on all object diameters, E_i and E_j must also be less than $2c \cdot \delta_{E_i}$ apart, which in turn implies that E_j must have non-empty intersection with the hypercube with side length $(4c+1) \cdot \delta_{E_i}$ having a smaller E_i-enclosing hypercube with side length δ_{E_i} at its center (see Figure 1 for a 2D example). Application of Lemma 2.2 yields that there can only be a constant number of such E_j's, so there is at most a constant number of wrapping boundaries $\partial\Delta(E_j)$ $(j > i)$ that intersect $\partial\Delta(E_i)$. By the additional assumption that all wrappings have constant complexity, there is only a constant number of subspaces of dimension between 0 and $d-1$ defined by the intersection of $\partial\Delta(E_i)$ and wrapping boundaries $\partial\Delta(E_j)$ $(j > i)$. Adding the contributions of all wrappings amounts to a total of $O(n)$ subspaces of dimensions 0 to $d-1$ in the arrangement $\mathcal{A}(\Delta)$. The linear bounds on the number of these subspaces imply the same bound of $O(n)$ on the number of d-faces in $\mathcal{A}(\Delta)$, making the total combinatorial complexity of the arrangement $O(n)$.

The (b)-part follows immediately from the proof of the (a)-part. Let E_i be the smallest object for which the point $p \in \mathbf{R}^d$ lies inside the wrapping $\Delta(E_i)$. Since there is only a constant number of wrappings of larger objects intersecting E_i's wrapping, the point p can be in no more than a constant number of additional wrappings. □

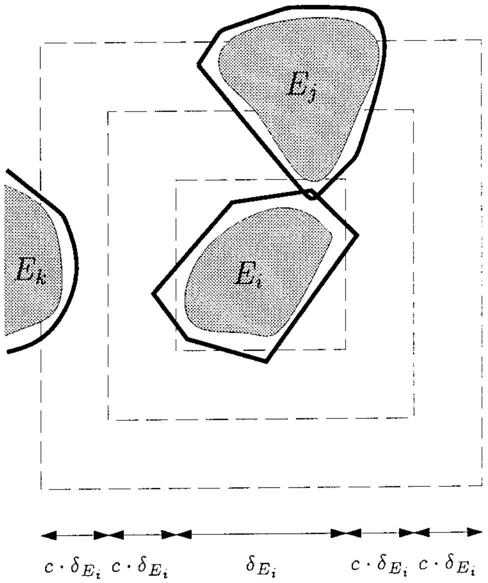


Figure 1: The object E_i can be fully enclosed in a box with side length δ_{E_i}. The $(c \cdot \delta)$-wrapping of E_i (with bold boundary) lies completely inside a concentric box with side length $(2c+1) \cdot \delta_{E_i}$, because $\delta \leq \delta_{E_i}$. A wrapping of some other obstacle E_j can only intersect the wrapping of E_i if it intersects the concentric box as well. By repeating the same reasoning, the object E_j itself must then intersect the outer concentric box with side length $(4c+1) \cdot \delta_{E_i}$.

3 Motion planning amidst fat obstacles

A number of cell decomposition-based motion planning algorithms (see e.g. [7, 8]) use a projection-like approach to partition the free space, i.e., they (recursively) decompose a lower-dimensional subspace of the configuration space C and lift the decomposition regions into C. The free part of the resulting cylinders is subsequently partitioned into a number of simple subcells. The objective is to keep the number of subcells as low as possible.

Our paradigm for motion planning amidst fat obstacles fits in the approach outlined above. The circumstances that we consider allow for a base decomposition in the workspace whose regions correspond to cylinders in configuration space in which the free part is easily decomposable into (a small number of) simple subcells. Below we first summarize the realistic assumptions that guarantee the feasibility of the approach.

We consider a not too large robot $\mathcal{B}$ with a constant number f of degrees of freedom moving in a d-dimensional Euclidean workspace amidst n k-fat obstacles $E \in \mathcal{E}$. Furthermore, we assume that no part of the robot $\mathcal{B}$ can collide with any other part of $\mathcal{B}$. Some point $O \in \mathcal{B}$ is appointed to be the reference point of the robot. Our paradigm applies to situations where the robot's configuration space C can be defined as the Cartesian product of its d-dimensional workspace W and some other space D (of dimension at least $f - d$),

$$C = \mathrm{W} \times D,$$

such that the position of the robot's reference point in the robot's workspace is part of the specification of its placement. A placement Z of the robot can thus be written as $Z = (Z_{\mathrm{W}}, Z_D)$, where $Z_{\mathrm{W}} \in \mathrm{W} = \mathbb{R}^d$ and $Z_D \in D$. In cases of a free-flying robot this assumption will be satisfied.

The reach $\rho_{\mathcal{B}}$ of a robot $\mathcal{B}$ is defined to be the maximum distance from the reference point $O \in \mathcal{B}$ to any point in $\mathcal{B}$ in any placement Z of $\mathcal{B}$. More formally:

Definition 3.1 [reach $\rho_{\mathcal{B}}$ of a robot $\mathcal{B}$]
Let Z_{W} be some arbitrary position of the reference point O of the robot $\mathcal{B}$. Then the reach $\rho_{\mathcal{B}}$ of the robot $\mathcal{B}$ is defined as

$$\rho_{\mathcal{B}} = \sup_{Z_D \in D} \max_{p \in \mathcal{B}[(Z_{\mathrm{W}}, Z_D)]} d(p, Z_{\mathrm{W}}),$$

where $\mathcal{B}[Z]$ stands for the set of points in W covered by $\mathcal{B}$ when placed at Z.

In other words, the reach $\rho_{\mathcal{B}}$ of a robot $\mathcal{B}$ is the maximum distance that any point in the robot $\mathcal{B}$ can ever have to the reference point, which is also equal to how far the robot can reach, measured from its reference point. Naturally, the reach is independent of the actual position of the reference point. Using the notion of the reach of a robot, we state the additional assumption that the robot is not too large: the reach $\rho_{\mathcal{B}}$ of the robot $\mathcal{B}$ should not exceed $b \cdot \delta_{min}$ for some reference point $O \in \mathcal{B}$ and some constant b, where δ_{min} is a lower bound on the diameter of the obstacles in $\mathcal{E}$.

If either the obstacles are non-fat or the robot is arbitrarily large, the robot $\mathcal{B}$ with its reference point O fixed at some point $p \in \mathrm{W}$ may be able to touch all obstacles $E \in \mathcal{E}$. The circumstances summarized above, however, make this impossible: the robot is able to touch only a constant number of obstacles. (This follows quite easily from Theorem 2.4.) As a result, the workspace can be partitioned into regions where the robot can only touch a constant number of obstacles. Note that $\mathcal{B}$ will definitely be unable to touch the obstacle E if it is further away from E than its reach $\rho_{\mathcal{B}}$. We define the notion of grown obstacles to formalize this observation.

Definition 3.2 [grown obstacle $G(E, \rho)$]
Let E be an obstacle in $\mathbb{R}^d$ and let $\rho \in \mathbb{R}^+$. The ρ-grown obstacle E is defined as:

$$G(E, \rho) = \{\, p \in \mathbb{R}^d \mid d(p, E) \leq \rho \,\}.$$

Clearly, $G(E, \rho_{\mathcal{B}})$ is a $\rho_{\mathcal{B}}$-wrapping of E. Now the robot's reference point must lie inside $G(E, \rho_{\mathcal{B}})$ in order for the robot $\mathcal{B}$ to be in contact with E. Because $\rho_{\mathcal{B}} \leq b \cdot \delta_{min}$, it follows from Theorem 2.4 that the arrangement $\mathcal{A}(G)$ of grown obstacle boundaries $\partial G(E, \rho_{\mathcal{B}})$ $(E \in \mathcal{E})$ has complexity $O(n)$ and each of the arrangement's d-faces is the intersection of only $O(1)$ grown obstacles.

If we take some arbitrary subset R of the workspace W, then the subset will intersect a number of grown obstacle regions. The set of obstacles for which the grown obstacle region intersects the set R will be referred to as R's coverage $Cov(R)$.

Definition 3.3 [coverage $Cov(R)$]
Let $R \subseteq \mathrm{W} = \mathbb{R}^d$.

$$Cov(R) = \{\, E \in \mathcal{E} \mid R \cap G(E, \rho_{\mathcal{B}}) \neq \emptyset \,\}.$$

A direct consequence of the conclusion that each d-face of $\mathcal{A}(G)$ is the intersection of only $O(1)$ grown obstacles is that the workspace W of the robot can be partitioned into regions R with $|Cov(R)| = O(1)$. In a general setting such a partitioning could be impossible as their may be points that lie in $\omega(1)$ grown obstacles simultaneously. Partitionings of the robot's workspace into constant complexity regions with constant size coverage provide a first step towards a decomposition of the free space into constant complexity subcells. We refer to workspace partitions of this kind as cc-partitions (constant-size coverage, constant complexity).

Definition 3.4 [cc-partition]
A cc-partition V of a workspace W with obstacles $\mathcal{E}$ is a partition of W into regions R satisfying the following additional constraints:

- $|Cov(R)| = O(1)$,
- *R has constant complexity.*

The cc-partition V_{W} of the workspace W can be lifted into a cylindrical decomposition of the configuration space C by taking the Cartesian product of the partition V_{W} and the space D. The specific properties of a region $R \in V_{\mathrm{W}}$, i.e., its constant complexity and constant-size coverage, are shown below to imply a constant descriptional complexity of the free part $\mathrm{FP} \cap (R \times D)$ of the configuration space cylinder $R \times D$. The maximal connected components of $\mathrm{FP} \cap (R \times D)$ are adequate subcells in a cell decomposition of the free space as they allow for easy path finding within their interior, due to their constant complexity and connectedness. Free subcells in a single configuration space cylinder $R \times D$ can only be adjacent to free subcells in a cylinder implied by a region R' that is adjacent to R in the cc-partition of the workspace W. As both subcells have constant complexity, their shared boundary must also have constant complexity and, as a result, allow for simple crossing from one subcell into another one.

Algorithm FatMot outlined below exploits the ideas summarized in the previous paragraph. The algorithm consists of two parts. The first part is a computation in the workspace that aims at finding a, hopefully small, cc-partition V_W of the workspace W with obstacles $\mathcal{E}$. More precisely, it computes a graph (V_W, E_W) consisting of a set V_W of closed regions and a set $E_W = \{(R, R') \in V_W \times V_W \,|\, \partial R \cap \partial R' \neq \emptyset\}$ of adjacencies of V_W's regions[1]. Finally, the graph must be accompanied by the computed function $Cov : V_W \rightarrow \mathcal{P}(\mathcal{E})$, linking the coverage $Cov(R)$ to each region $R \in V_W$. The actual "shape" of the first part of Algorithm FatMot depends on the specific workspace W and type of obstacles in $\mathcal{E}$ under consideration, as does the computation time $T(n)$ that is spent in this phase. The computation time $T(n)$ turns out to highly affect the efficiency of the entire Algorithm FatMot, hence, a thorough investigation of instances of W and $\mathcal{E}$ is justified. This is the subject of the next section.

The remaining steps of Algorithm FatMot take place in configuration space, where the cc-partition graph (V_W, E_W) and the function Cov are used to deduce a connectivity graph $CG = (V_C, E_C)$ consisting of a set of constant complexity regions V_C that partition the set of free placements FP, and the set $E_C = \{(A, A') \in V_C \times V_C \,|\, \partial A \cap \partial A' \neq \emptyset\}$. The sizes of the sets V_C and E_C are of the same order of magnitude as the sets V_W and E_W respectively: $|V_C| = O(|V_W|)$, $|E_C| = O(|E_W|)$. The outline of the second part is, contrary to the first part, independent of the workspace W and obstacles $\mathcal{E}$ under consideration.

ALGORITHM FATMOT

```
Find a cc-partition graph $(V_W, E_W)$ and compute $Cov$;
$V_C := \emptyset$;
$E_C := \emptyset$;
for all $R \in V_W$ do
    1. $Desc(R) := \emptyset$;
    2. for all $E \in Cov(R)$ do
           for all features $\phi$ of $\mathcal{B}$ and $\Phi$ of $E$ do
               compute constraint surface $f_{\phi,\Phi}$;
    3. compute the arrangement $\mathcal{A}$ of surfaces $f_{\phi,\Phi}$;
    4. use $\mathcal{A}$ to compute FP $\cap (R \times D)$;
    5. for all maximal connected components $A$
       of FP $\cap (R \times D)$ do
           5.1. $V_C := V_C \cup \{A\}$;
           5.2. $Desc(R) := Desc(R) \cup \{A\}$;
for all $(R_1, R_2) \in E_W$ do
    for all $A_1 \in Desc(R_1) \wedge A_2 \in Desc(R_2)$ do
        if $\partial A_1 \cap \partial A_2 \neq \emptyset$ then $E_C := E_C \cup \{(A_1, A_2)\}$.
```

Figure 2 gives a pictorial explanation of Algorithm FatMot.

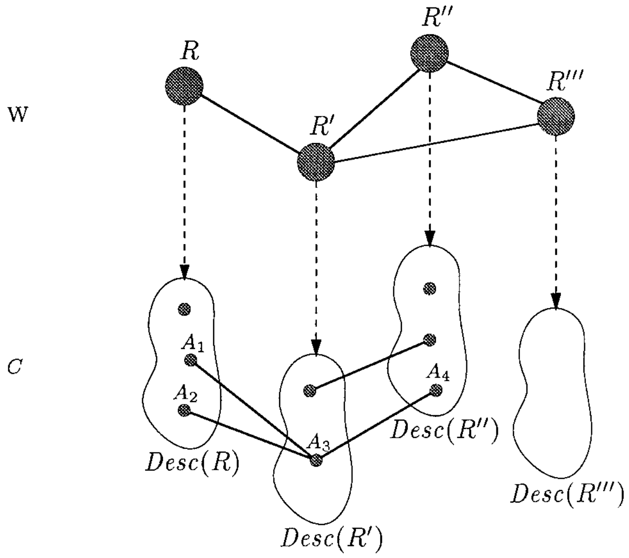


Figure 2: The relation between the cc-partition graph (V_W, E_W) in the workspace W at the top, and the connectivity graph (V_C, E_C) in the configuration space at the bottom. Each node $R \in V_W$ defines at most $O(1)$ nodes $A \in V_C$, collected in a set $Desc(R)$. Two nodes A and A' in V_C can only be connected if the corresponding nodes R and R' in V_W are connected, so A_1 may be connected to all nodes in $Desc(R')$, but A_1 and A_4 cannot be connected.

Crucial to the validity of the approach sketched by Algorithm FatMot is a rather simple observation regarding constraint surfaces in configuration space. Let us consider the surface $f_{\phi,\Phi}$ of placements in which a robot feature ϕ touches a feature Φ of some obstacle E. A substantial part of the representation of a robot placement is the position, in the workspace W, of the robot's reference point. This position must be less than $\rho_{\mathcal{B}}$ away from E, otherwise $\mathcal{B}$ would be unable to touch E. As a result, a point $p = (p_W, p_C) \in W \times D$ lying on the constraint surface $f_{\phi,\Phi}$ satisfies $p_W \in G(E, \rho_{\mathcal{B}})$.

Lemma 3.5 *Let ϕ a robot feature and let Φ be a feature of an obstacle E, such that the placements in which ϕ is in contact with Φ define a hypersurface $f_{\phi,\Phi}$ in C, then:*

$$f_{\phi,\Phi} \subseteq G(E, \rho_{\mathcal{B}}) \times D.$$

An implication of Lemma 3.5 is that a configuration space cylinder $R \times D$ can only be intersected by constraint surfaces $f_{\phi,\Phi}$, for which Φ is a feature of some $E \in Cov(R)$. By the constant-size coverage of a region $R \in V_W$ and the constant complexity of both $\mathcal{B}$ and each obstacle $E \in \mathcal{E}$, the number of constraint surfaces $f_{\phi,\Phi}$ intersecting a cylinder $R \times D$ with $R \in V_W$, is bounded by a constant. Since each constraint surface is assumed to have constant complexity,

[1] We do not distinguish between a node in the graph and the region that is represented by that node, hence if we speak of a node R (or A) we actually mean the node representing the region R (or the free subcell A).

the entire arrangement of constraint surfaces inside $R \times D$ has constant complexity. Together with the constant complexity of the cylinder $R \times D$ itself - following from R's constant complexity - this implies the constant complexity of $\mathrm{FP} \cap (R \times D)$. Steps 1-5 in the first **for**-loop compute the $O(1)$ (constant complexity) maximal connected components of $\mathrm{FP} \cap (R \times D)$. A possible way to compute this free part (in constant time) could be to apply the techniques from [8] to the constant number of constraint hypersurfaces intersecting the cylinder $R \times D$. Each step is easily verified to run in $O(1)$ time. The set $Desc(R)$ stores the nodes in (V_C, E_C) that correspond to free subcells in $R \times D$. Note that the set $Desc(R)$ has constant cardinality.

Two free subcells A_1 and A_2 can only be adjacent if their containing cylinders $R_1 \times D$ and $R_2 \times D$ are adjacent in C and, hence, R_1 and R_2 are adjacent in W. An adjacency (R_1, R_2) gives rise to only a constant number of adjacencies of nodes A_1 and A_2 in $Desc(R_1)$ and $Desc(R_2)$ respectively due to the constant cardinality of $Desc(R_1)$ and $Desc(R_2)$. Two free subcells A_1 and A_2 in adjacent cylinders are adjacent if they share a common boundary. Such a common boundary has constant complexity since both involved free subcells have constant complexity. The nested **for**-loop in the second **for**-loop takes constant time by the above considerations.

The output of Algorithm FatMot is a decomposition of the free space into constant complexity subcells that are collected in a set V_C. The connectivity of the subcells is captured in a set E_C of pairs of adjacent subcells. The running time of FatMot is easily seen to be $O(T(n) + |V_{\mathrm{W}}| + |E_{\mathrm{W}}|)$. So, the efficiency of Algorithm FatMot is fully determined by the size of the cc-partition graph $(V_{\mathrm{W}}, E_{\mathrm{W}})$ and the time to compute the graph and the function Cov. Since the time $T(n)$ to explicitly compute a cc-partition graph $(V_{\mathrm{W}}, E_{\mathrm{W}})$ and the function Cov dominates the time $O(|V_{\mathrm{W}}| + |E_{\mathrm{W}}|)$ to simply report the graph we may conclude that the $T(n)$-factor is dominant.

Theorem 3.6 *Algorithm FatMot computes a cell decomposition of the free space of size* $O(|V_{\mathrm{W}}| + |E_{\mathrm{W}}|)$ *in time* $O(T(n))$, *where* $T(n)$ *is the time to compute a cc-partition graph* $(V_{\mathrm{W}}, E_{\mathrm{W}})$ *and the function* $Cov : V_{\mathrm{W}} \to \mathcal{P}(\mathcal{E})$.

The size and computing time of a cc-partition graph influence both the size of the cell decomposition that is obtained and the time necessary to compute it. A small and efficiently computable partition is therefore crucial to the success of our method. At this stage, one may expect the method to be rather efficient since the paradigm reduces the problem of finding a cell decomposition of the free part of the configuration space to the problem of finding some constrained partition of the lower-dimensional workspace. Finding small and efficiently computable cc-partitions is the subject of the next section.

Finally, we mention that the problem of solving a motion planning query "find a free path from a placement $Z_1 = (Z_{1\mathrm{W}}, Z_{1D})$ to another placement $Z_2 = (Z_{2\mathrm{W}}, Z_{2D})$" basically reduces to a point location query with $Z_{1\mathrm{W}}$ and $Z_{2\mathrm{W}}$ in V_{W} to find $R_1 \ni Z_{1\mathrm{W}}$ and $R_2 \ni Z_{2\mathrm{W}}$. Hence, only point location in the (2D or 3D) workspace is required, which can be done efficiently. After that it takes $O(1)$ to find $A_1 \ni Z_1$ using $Desc(R_1)$ and $A_2 \ni Z_2$ using $Desc(R_2)$, followed by a search in the connectivity graph (V_C, E_C) for a sequence of subcells connecting A_1 to A_2. The constant complexities of the subcells and of the common boundaries of pairs of adjacent subcells facilitate the transformation of the subcell sequence into an actual free path for $\mathcal{B}$.

4 Small and efficiently computable workspace partitions

In this section we consider the problem of finding small cc-partitions of workspaces W with obstacles $\mathcal{E}$. Different types of workspaces and obstacles lead to different choices for a cc-partition. Below we first discuss a general partition scheme for a 2D Euclidean workspace with constant complexity obstacles. The cc-partition has optimal size and is computable by a simple plane sweep. Next, we move on to 3D, where we consider the case of polyhedral obstacles. In addition, we mention some results obtained for general obstacles and obstacles from a bounded size range.

4.1 Two-dimensional Euclidean workspace

The proposed partition scheme for 2D workspaces, a vertical decomposition of the arrangement of grown obstacle boundaries, works for any kind of obstacles, as long as their boundaries consist of a constant number of constant degree algebraic curves. The approach differs from the one presented in [13] which is a modified version of Schwartz and Sharir's ladder algorithm [7] and as such dedicated to a polygonal robot and polygonal obstacles.

The first step towards a cc-partition computes the grown obstacle boundaries $\partial G(E, \rho_{\mathcal{B}})$ for all obstacles $E \in \mathcal{E}$. Each boundary is obtained in $O(1)$ time leading to $O(n)$ time for computing all boundaries. As a preparation for the next step, each grown obstacle boundary $\partial G(E, \rho_{\mathcal{B}})$ is cut up into $O(1)$ arcs which are maximal connected, x-monotonous boundary parts having no vertices in their interiors. Note that arc endpoints are generally incident to two arcs. For future purposes we label each arc from $\partial G(E, \rho_{\mathcal{B}})$ with the obstacle name 'E'. By Theorem 2.4, the resulting $O(n)$ arcs define only $O(n)$ (yet unknown) arc intersections and partition the workspace into regions with constant-size coverage.

In a second step we compute the vertical decomposition of the arrangement $\mathcal{A}(G)$ of grown obstacle boundaries, by sweeping the plane [3] with the arcs with a vertical line, meanwhile extending walls in upward and downward vertical direction from every arc endpoint (known in advance) and all arc intersections (to be determined during the sweep). The arcs and walls and their endpoints form a planar graph consisting of $O(n)$ edges and vertices, subdividing the plane into $O(n)$ constant complexity regions: the regions of V_{W}. The set E_{W} of pairs of adjacent regions has size $O(n)$ as well, as each adjacency is represented by one of the $O(n)$ edges in the graph.

Lemma 4.1 $|V_{\mathrm{W}}| = O(n)$ *and* $|E_{\mathrm{W}}| = O(n)$.

The sweep must not only compute the regions of V_{W}, but also the coverages $Cov(R)$ of the regions $R \in V_{\mathrm{W}}$, and the region adjacencies of E_{W}. Throughout the sweep we maintain as invariant that all regions, coverages, and adjacencies left of the sweep-line are computed. The sweep-line status and event point schedule, both stored in appropriate data structures, facilitate the maintenance of the invariant. The sweep-line status is a top-to-bottom cross-section of the vertical decomposition of $\mathcal{A}(G)$ and the vertical sweep-line. This alternating sequence of arcs and regions accompanied by their coverages is stored in a binary tree. The event point schedule is the ordered sequence (by x-coordinate) of all arc endpoints and the potential intersections of consecutive arcs in the sweep-line status. A priority queue stores the schedule.

The processing of the next event, i.e., the extension of vertical walls from an intersection point or endpoint, marks the end of at most three consecutive regions in the sweep-line status. These regions and their coverages are reported to maintain the invariant. Moreover, the regions are deleted from the sweep-line status and replaced by the (at most three) newly started regions. Appropriate adjustments w.r.t. the separating arcs are made as well. In addition, we must report the adjacencies of the new regions and compute their coverages. These coverages are easily computed from the coverages of the ended regions and the arc labels (see [14] for details on the update). Potential intersections of the $O(1)$ newly created pairs of consecutive arcs must be inserted in the event queue. In summary, the processing of an event requires a constant number of searches, deletions, and insertions on a binary tree, and a constant number of insertions in the event queue. All listed operations take $O(\log n)$ time, so the processing of all events, and, hence, the entire sweep, takes $O(n \log n)$ time.

Lemma 4.2 $T(n) = O(n \log n)$.

Substitution of the vertical decomposition in the first step of Algorithm FatMot leads to a cell decomposition of size $O(n)$, computed in $O(n \log n)$ time.

Theorem 4.3 *The motion planning problem amidst fat obstacles in* $\mathbf{R}^2$ *can be solved in time* $O(n \log n)$.

4.2 Three-dimensional Euclidean workspace

The problem of finding cc-partitions for three-dimensional Euclidean workspaces is much harder than its two-dimensional equivalent, which is illustrated by the relatively small number of results on partitioning 3D arrangements. Below we first consider the case of polyhedral obstacles in some detail, and then mention some results for two other classes of obstacles.

Like in the two-dimensional case, we first partition the workspace into regions with constant-size coverage, and subsequently subdivide these regions into constant complexity regions. Instead of using the grown obstacle boundaries to achieve the first decomposition we now use the boundary of a polyhedral outer approximation of these grown obstacles, which still achieves a decomposition of W into regions of constant-size coverage, but additionally allows for subsequent application of a vertical trapezoidation algorithm (to the triangulated polyhedral arrangement) to obtain $O(n^2)$ constant complexity regions. A tight outer approximation of the grown obstacle $G(E, \rho_{\mathcal{B}})$ is the Minkowski difference $H(E, \rho_{\mathcal{B}})$ of E and a cube with side length $2\rho_{\mathcal{B}}$:

$$H(E, \rho_{\mathcal{B}}) = \\ E \ominus (\Theta \cdot \{(x_1, x_2, x_3) \in \mathbf{R}^3 | -\rho_{\mathcal{B}} \leq x_1, x_2, x_3 \leq \rho_{\mathcal{B}}\}),$$

where $\ominus$ represents the Minkowski difference operator and $\Theta \in SO(3)$ is some arbitrary rotation matrix to establish that none of the cube's faces is vertical, thus preventing future vertical triangles. Computation of a (constant complexity) Minkowski difference $H(E, \rho_{\mathcal{B}})$ takes $O(1)$ time.

The polyhedral arrangement $\mathcal{A}(H)$ of all Minkowski difference boundaries $\partial H(E, \rho_{\mathcal{B}})$, $E \in \mathcal{E}$, satisfies properties that are similar to the properties of the arrangement $\mathcal{A}(G)$ in the 2D case.

Lemma 4.4 *(a) The complexity of* $\mathcal{A}(H)$ *is* $O(n)$,

(b) $|Cov(A)| = O(1)$ *for each 3-face* A *in* $\mathcal{A}(H)$.

Proof: Each Minkowski difference $H(E, \rho_{\mathcal{B}})$ is a $\sqrt{3} \cdot \rho_{\mathcal{B}}$-wrapping of E, and, because $\rho_{\mathcal{B}} \leq b \cdot \delta_{min}$, also a $b\sqrt{3} \cdot \delta_{min}$-wrapping of E. Since δ_{min} is a lower bound on the diameter of all obstacles $E \in \mathcal{E}$, the (a)-part follows immediately from Theorem 2.4. Moreover, the same theorem implies that each point $p \in$ W lies in $O(1)$ Minkowski differences $H(E, \rho_{\mathcal{B}})$ simultaneously, which implies that a 3-face A lies in the intersection of $O(1)$ Minkowski differences. Because $H(E, \rho_{\mathcal{B}}) \supseteq G(E, \rho_{\mathcal{B}})$, $Cov(A)$ is a subset of the set of obstacles E for which $H(E, \rho_{\mathcal{B}})$ encloses A and, hence, $|Cov(A)| = O(1)$. □

A naive, but sufficiently efficient, computation of the $O(n)$ complexity arrangement $\mathcal{A}(H)$ takes $O(n^2)$ time and a subsequent constrained triangulation of all $O(n)$ planar Minkowski faces and the segments defined by intersections with other faces can be performed in near-linear time. The constrained triangulation of the faces of $\mathcal{A}(H)$ does not introduce new vertices. The result is that we end up with a collection $T_{\mathcal{A}(H)}$ of non-intersecting triangles. Although the triangles are non-intersecting, they do touch each other, i.e., they share edges and vertices.

De Berg, Guibas, and Halperin [4] present an algorithm for computing the vertical trapezoidation of an arrangement of triangles in 3-space. In the restricted case of non-intersecting triangles, their algorithm leads to a decomposition of 3-space into $O(n^2)$ constant complexity regions. The computation takes $O(n^2 \log n)$ time. Below we first briefly discuss the algorithm and the structure of the trapezoidation. After that, we show that application of the algorithm to the set of triangles $T_{\mathcal{A}(H)}$ leads to a cc-partition graph $(V_{\mathrm{W}}, E_{\mathrm{W}})$, with $|V_{\mathrm{W}}| = O(n^2)$ but, unfortunately, a larger set E_{W}. To remedy this we will replace each triangle by a flat tetrahedron and then show that the resulting set $F(T_{\mathcal{A}(H)})$ of tetrahedra solves the problem as its vertical trapezoidation leads to $|V_{\mathrm{W}}| = |E_{\mathrm{W}}| = O(n^2)$ and $T(n) = O(n^2 \log n)$.

The computation of the vertical trapezoidation proceeds in two steps. The first step results in the vertical decomposition of the arrangement of non-intersecting triangles. The vertical decomposition partitions the arrangement into regions of points with equal vertical visibility w.r.t. T, both in upward and downward direction, by extending walls in vertical direction from all triangle edges. A region in the vertical decomposition is a maximal connected set $\{x \in \mathbf{R}^3 | up(x) = t_1 \wedge down(x) = t_2\}$ where $t_1, t_2 \in T$ and $up(x)$/$down(x)$ denotes the first triangle in T that is hit by the vertical ray emanating from x in upward/downward direction. The floors and ceilings, i.e., the bounding faces in vertical direction, of the resulting regions are parts of triangles. More precisely, they are 2-faces in an arrangement of ending walls on a triangle side. Note that the floor and ceiling of a vertical decomposition region have equivalent projections onto the (x, y)-plane. The second step refines the vertical decomposition into a vertical trapezoidation, consisting of regions bounded by six planar faces, by adding walls parallel to the (x, z)-plane. The refining is obtained through a planar vertical decomposition of the floor and ceiling of the region, in which segments are extended within the floor or ceiling parallel to the y-axis from every vertex on the boundary of the floor or ceiling. These segments delimit the additional walls parallel to the (x, z)-plane. The resulting trapezoidal faces are the floors and ceilings of the vertical trapezoidation regions.

The vertical trapezoidation algorithm results in $O(n^2)$ constant complexity regions in time $O(n^2 \log n)$. The output consists of the input triangles accompanied by the vertically decomposed arrangements of ending walls. These arrangements implicitly contain all relevant information regarding the vertical trapezoidation. The representation of a region follows immediately from its floor and ceiling. Adjacent regions in the trapezoidation share a common boundary that is either embedded in a vertical wall, either extended from a triangle edge or added during the trapezoidation phase, or embedded in a triangle. If the triangles are in general position then it can be shown that regions that share a vertical boundary face must have either adjacent floors or adjacent ceilings in some vertically decomposed triangle arrangement. Regions that share a common boundary that is embedded in a triangle, have floor and ceiling with non-empty intersection on either side of a triangle. Hence, the representation of the regions as well as the adjacency information is completely contained in the vertically decomposed triangle arrangements.

Adjacent regions in the decomposition share a face that is either a part of a vertical wall or a part of some triangle. It turns out that the number of adjacencies of the latter type is more than $O(n^2)$. Walls extended from other triangles' edges end on both sides of a triangle $t \in T_{\mathcal{A}(H)}$ and define arrangements of line segments on these sides. The complexity of a single arrangement and its vertical decomposition can be as high as $O(n^2)$, and, hence, the number of 2-faces defined by the vertically decomposed arrangement is bounded by $O(n^2)$. These 2-faces are the floors (or ceilings) of the vertical trapezoidation regions. Each non-empty intersection of 2-faces on either side of a single triangle corresponds to an adjacency of two vertical trapezoidation regions. There are $O(n^4)$ pairs of 2-faces on opposite sides of a triangle and it seems impossible to bound the number of non-empty intersections, and, hence, the number of region adjacencies, by anything close to $O(n^2)$.

An elegant way to avoid the problem outlined above is by replacing the set of triangles $T_{\mathcal{A}(H)}$ by a set of tetrahedra $F(T_{\mathcal{A}(H)})$ that are sufficiently flat to prevent them from intersecting. The tetrahedra have the initial triangles of $T_{\mathcal{A}(H)}$ as one of their faces. The triangular faces of the tetrahedra in $F(T_{\mathcal{A}(H)})$ form again a set of non-intersecting, though touching, triangles. This set is four times larger than $T_{\mathcal{A}(H)}$ (and therefore still of size $O(n)$) but, more importantly, it has the simple though very beneficial property that one of the sides of each triangle faces the interior of a tetrahedron. As a result, the number of walls ending on the interior-facing side of a triangular face t of $\tau \in F(T_{\mathcal{A}(H)})$ is constant as such walls can only be extended from edges of the tetrahedron τ.

We first discuss how to replace each triangle $t \in T_{\mathcal{A}(H)}$ by a tetrahedron τ having t as one of its faces, such that the resulting tetrahedra are non-intersecting. Recall that $T_{\mathcal{A}(H)}$ is a set of triangles that may share a vertex or an edge but do not intersect each others' interiors. Let ϵ be the minimum distance between any pair of disjoint triangles. Furthermore, let γ_1 be the minimum over all dihedral angles between pairs of (touching) triangles that share an edge and γ_2 be the minimum over all angles between the supporting plane of a triangle $t \in T_{\mathcal{A}(H)}$ and an edge of another triangle t' incident to a vertex of t. We define $\gamma = \min(\gamma_1, \gamma_2)$. We construct a set $F(T_{\mathcal{A}(H)})$ by applying the following procedure to every $t \in T$. Let v_1, v_2, v_3 be the vertices of t.

1. The planes π_1, π_2, π_3 through v_1, v_2, v_3 that make a positive angle $\gamma/2$ with the top side (facing $z = \infty$) of t intersect in a point v. If the distance from v to t is strictly less than ϵ, then the tetrahedron is defined by the vertices v, v_1, v_2, v_3.
2. If the distance from v to t is at least ϵ, then we take the half-line h through v and perpendicular to and ending on t. The tetrahedron is defined by v_1, v_2, v_3 and the unique point v' on the half-line h with distance $\epsilon/2$ to t.

The following lemma now holds for the resulting set $F(T_{\mathcal{A}(H)})$ of tetrahedra. (For a proof, we refer to [14].) We denote the (open) interior of a closed set τ by $int(\tau)$.

Lemma 4.5 $\forall \tau, \tau' \in F(T_{\mathcal{A}(H)}) : int(\tau) \cap int(\tau') = \emptyset$.

Application of the trapezoidation algorithm to the triangular faces of $F(T_{\mathcal{A}(H)})$ yields a vertical trapezoidation of complexity $O(n^2)$ and therefore consisting of $O(n^2)$ regions with constant complexity. The trapezoidation regions are appointed to be the regions of V_W. The coverage of each region $R \in V_W$ clearly has constant size as R is a subset of a 3-face A in the arrangement $\mathcal{A}(H)$.

Let us now consider the number of adjacencies of regions in V_W. The complexities of the vertically decomposed triangle arrangements is crucial in determining the number of adjacencies, so we first study these complexities. The complexity of the entire vertical trapezoidation is $O(n^2)$. As a result the cumulative complexity of all vertically decomposed triangle arrangements is $O(n^2)$ as well. Each triangle $t \in F(T_{\mathcal{A}(H)})$ has a side facing the interior of the tetrahedron τ it belongs to and a side facing outward. The complexity m_t of the vertically decomposed arrangement on the inward-facing side of t is constant, since only a constant number of walls, extended from one of the edges of τ, ends on this side: $m_t = O(1)$, for all t. The complexity n_t of the vertically decomposed arrangement on the outward-facing side of a single triangle t, however, can be as high as $O(n^2)$. The number of adjacencies of 2-faces in a decomposed triangle arrangement is of the same order of magnitude as the complexity of the decomposed arrangement, hence, the numbers of adjacencies on the inward and outward-facing sides of a triangle t are $O(m_t)$ and $O(n_t)$ respectively, adding up to a total of $O(m_t + n_t)$ adjacencies of 2-faces on t. We have stressed earlier that each pair of adjacent trapezoidation regions has either adjacent floors or ceilings in some vertically decomposed triangle arrangement, or intersecting floor and ceiling on opposite triangle sides. The number of adjacencies of the first type equals $\sum_t O(m_t + n_t) = O(n^2)$, as it is of the same order of magnitude as the cumulative complexity of the vertically decomposed triangle arrangements. The number of non-empty intersections on opposite sides of a triangle t is $O(m_t \cdot n_t) = O(n_t)$ due to $m_t = O(1)$. Hence, the number of adjacencies of the second type equals $\sum_t O(n_t) = O(n^2)$. In summary, the total number of adjacencies in E_W is $O(n^2)$.

Lemma 4.6 $|V_W| = O(n^2)$ *and* $|E_W| = O(n^2)$.

After applying the $O(n^2 \log n)$ trapezoidation algorithm, we traverse the resulting $O(n^2)$ constant complexity regions using the decomposed triangle arrangements to extract explicit descriptions of the regions, regions adjacencies, and the coverages of the regions. The latter computation requires additional explanation. Instead of attempting to compute the constant-size coverages directly, we first compute the set of intersecting Minkowski differences $H(E, \rho_{\mathcal{B}})$, using the fact (throughout the traversal) that adjacent regions are intersected by the same set of Minkowski difference unless they are separated by a triangle t that is part of some Minkowski difference boundary $\partial H(E, \rho_{\mathcal{B}})$, in which case the sets of intersecting Minkowski differences differ by exactly $\{E\}$. The set of Minkowski differences $H(E, \rho_{\mathcal{B}})$ intersecting a region $R \in V_W$ is a superset of the set of grown obstacles $G(E, \rho_{\mathcal{B}})$ intersecting R, since $G(E, \rho_{\mathcal{B}}) \subseteq H(E, \rho_{\mathcal{B}})$. The latter set, which is the coverage $Cov(R)$ of R can be computed from the set of intersecting Minkowski differences, as this set has constant size as well by Lemma 4.4. The traversal of the $O(n^2)$ regions takes time proportional to the number of regions, leading to the conclusion that the computation time $T(n)$ is dominated by the running time of the trapezoidation algorithm.

Lemma 4.7 $T(n) = O(n^2 \log n)$.

Substitution of the described vertical trapezoidation in the first step of Algorithm FatMot leads to a cell decomposition of size $O(n^2)$ computed in $O(n^2 \log n)$ time.

Theorem 4.8 *The motion planning algorithm amidst fat polyhedral obstacles in* $\mathbf{R}^3$ *can be solved in time* $O(n^2 \log n)$.

If the assumption of polyhedral obstacles is dropped we are no longer able to tightly wrap the obstacles. As a consequence, we are also no longer able to obtain an arrangement of wrapping boundaries of size $O(n)$ that partitions the workspace into regions of constant-size coverage. Hence, a different strategy for finding a cc-partition is necessary. It can be shown [14] that a possible cc-partition of the workspace is given by the supporting planes of the faces of the bounding boxes of the grown obstacles $G(E, \rho_{\mathcal{B}})$. This partition yields $O(n^3)$ regions and adjacencies, and leads to an $O(n^3)$ motion planning algorithm, which seems definitely open for improvement.

Another nice result is obtained for general obstacles from a bounded size range. If we assume all obstacle diameters to be in the range $\delta_{min} \ldots \delta_{max} = u \cdot \delta_{min}$ for some constant $u > 1$, then a cc-partition of optimal $O(n)$ size can be constructed. The idea is to put an imaginary orthogonal grid $\{(p_1\delta_{min}, p_2\delta_{min}, p_3\delta_{min}) | p_1, p_2, p_3 \in \mathbf{Z}\}$ in the workspace W and partition W into axis-parallel boxes with the grid points as their corners. A key observation is that the bounding box of each grown obstacle can be covered with a constant number of grid cubes with side length δ_{min}, leading to $O(n)$ such cubes with constant size coverage by Lemma 2.2. The complement of the cubes is decomposable by $O(n)$ axis-parallel boxes of measure $\delta_{min} \times \delta_{min} \times h \cdot \delta_{min}$, with $h \in \mathbf{N}$, and $O(n)$ axis-parallel boxes that are unbounded in the z-direction. The resulting regions provide a cc-partition graph $(V_{\mathrm{W}}, E_{\mathrm{W}})$ of size $O(n)$, that can be computed together with the region coverage Cov in time $O(n \log n)$ (see again [14] for the details).

Theorem 4.9 *The motion planning problem amidst fat obstacles in* $\mathbb{R}^3$ *with diameters in the range* $\delta_{min} \ldots u \cdot \delta_{min}$, *for some constant* $u > 1$, *can be solved in time* $O(n \log n)$.

5 Conclusion

In this paper, we have presented a paradigm for planning the motion of a constant complexity robot amidst fat constant complexity obstacles. The efficiency of the method relies on the ability to find small and efficiently computable so-called cc-partitions of the workspace. We have furthermore shown that small cc-partitions of the workspace do indeed exist for various instances of workspaces with obstacles. A cc-partition of optimal size, i.e., $O(n)$, has been found for planar Euclidean workspaces with arbitrarily shaped obstacles and for 3D Euclidean workspaces with arbitrary obstacles of comparable sizes. The partitions, both computable in $O(n \log n)$, lead to nearly-optimal algorithms for motion planning in the corresponding settings. Additional cc-partitions have been found for Euclidean workspaces with polyhedral and arbitrary obstacles. They lead to motion planning algorithms in the respective settings with running times $O(n^2 \log n)$ and $O(n^3)$, based on quadratic and cubic size partitions. An obvious open question is whether it is possible to enhance the latter results by finding smaller, efficiently computable, cc-partitions for these environment instances.

The presented paradigm has a large applicability. The listed bounds and algorithms are independent of the actual number of degrees of freedom of the robot, as long as this number is constant. The paradigm therefore not only caters for rigid robots but is instead also capable of handling robots that both move and have articulated arms and legs. Moreover, preliminary results on using (a slightly generalized version of) the paradigm for planning coordinated motions of two or more robots amidst fat obstacles are promising.

References

[1] H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig, Approximate motion planning and the complexity of the boundary of the union of simple geometric figures, *Algorithmica* **8** (1992), pp. 391-406.

[2] F. Avnaim, J.-D. Boissonnat, and B. Faverjon, A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles, *Proc. Geometry and Robotics Workshop* (J.-D. Boissonnat and J.-P. Laumond Eds.), Lecture Notes in Computer Science **391** (1988), pp. 67-86.

[3] J.L. Bentley and T.A. Ottman, Algorithms for reporting and counting geometric intersections, *IEEE Transactions on Computers* **28** (1979), pp. 643-647.

[4] M. de Berg, L.J. Guibas, and D. Halperin, Vertical decompositions for triangles in 3-space, *Proc. 10th ACM Symp. on Computational Geometry* (1994).

[5] J. Matoušek, N. Miller, J. Pach, M. Sharir, S. Sifrony, and E. Welzl, Fat triangles determine linearly many holes, *Proc. 32nd IEEE Symp. on Foundations of Computer Science* (1991), pp. 49-58.

[6] M.H. Overmars, Point location in fat subdivisions, *Inform. Proc. Lett.* **44** (1992), pp. 261-265.

[7] J.T. Schwartz and M. Sharir, On the piano movers' problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal boundaries, *Comm. Pure Appl. Math.* **36** (1983), pp. 345-398.

[8] J.T. Schwartz and M. Sharir, On the piano movers' problem II. General techniques for computing topological properties of real algebraic manifolds, *Adv. in Applied Mathematics* **4** (1983), pp. 298-351.

[9] J.T.Schwartz and M. Sharir, Efficient motion planning algorithms in environments of bounded local complexity, Report 164, Department of Computer Science, Courant Inst. Math. Sci., New York NY (1985).

[10] S. Sifrony and M. Sharir, A new efficient motion planning algorithm for a rod in two-dimensional polygonal space, *Algorithmica* **2** (1987), pp. 367-402.

[11] A.F. van der Stappen, D. Halperin, and M.H. Overmars, The complexity of the free space for a robot moving amidst fat obstacles, *Computational Geometry: Theory and Applications*, **3** (1993), pp. 353-373.

[12] A.F. van der Stappen, The complexity of the free space for motion planning amidst fat obstacles, *Journal of Intelligent & Robotic Systems*, in press.

[13] A.F. van der Stappen, D. Halperin, and M.H. Overmars, Efficient algorithms for exact motion planning amidst fat obstacles, *Proc. of the IEEE Int. Conf. on Robotics and Automation*, Atlanta GA (1993), Vol. 1, pp. 297-304.

[14] A.F. van der Stappen and M.H. Overmars, Motion planning amidst fat obstacles, in preparation.

Approximate Euclidean Shortest Path in 3-Space

(Extended Abstract)

Joonsoo Choi Jürgen Sellen* Chee-Keng Yap†
Courant Institute of Mathematical Sciences
New York University

Abstract

Papadimitriou's approximation approach to the Euclidean shortest path (ESP) problem in 3-space is revisited. As this problem is NP-hard, his approach represents an important step towards practical algorithms. Unfortunately, there are non-trivial gaps in the original description. Besides giving a complete treatment, we also give an alternative to his subdivision method which has some nice properties. Among the tools needed are root-separation bounds and non-trivial applications of Brent's complexity bounds on evaluation of elementary functions using floating point numbers.

1 Introduction

The *Euclidean shortest path* (ESP) problem can be formulated as follows: given a collection of polyhedral obstacles in physical space S, and source and target points $s_0, t_0 \in S$, find a shortest obstacle-avoiding path between s_0 and t_0. Here S is typically $\mathbb{E}^2$ or $\mathbb{E}^3$. It is evident that this is a basic problem in applications such as robotics. The case $S = \mathbb{E}^2$ has recently seen a breakthrough, with the $O(n \log n)$ algorithm of Suri and Hershberger [7]. However, our interest is the case $S = \mathbb{E}^3$. Here, Canny and Reif [2] proved that the problem is NP-hard. In contrast, Papadimitriou [6] described a polynomial-time approximate algorithm. The *approximate Euclidean shortest path* problem has, in addition to the above inputs, an input parameter $\epsilon > 0$ and we are required to construct an obstacle-avoiding path whose length is at most $(1+\epsilon)$ times the length of the shortest path. As Papadimitriou suggested, his algorithm offers hope for practical implementation. The approach is basically simple and may be very similar to ones that practitioners intuitively use, although without a corresponding analysis. Unfortunately, there are several non-trivial gaps in this important paper. There is an alternative approach to the approximate ESP problem due to Clarkson [3]; but it is not directly comparable to Papadimitriou's result because the complexity of Clarkson's algorithm depends on an additional parameter ρ.

Most geometric algorithms are developed within the context of one of two distinct computational frameworks, the *algebraic framework* or the *bit framework*. In the algebraic framework, we count the number of algebraic operations, assuming that each algebraic operation returns an exact value. Computational models appropriate for the algebraic framework include arithmetic circuits and real RAMS. In the bit framework, we are interested in the representation of algebraic values as binary strings. Algebraic operations must be reduced to Boolean operations. Computational models that cater to the bit framework include Boolean circuits and Turing machines.

Most basic problems in the algebraic framework have only one input size parameter n, which is usually proportional to the number of algebraic values in the input. For instance, the problem of multiplying two polynomials of degree n is said to have input size n, and can be solved in $O(n \log n)$ algebraic operations. The result of Suri and Hershberger mentioned above also falls under the algebraic framework. Basic problems in the bit framework usually have two parameters, n and L, where n is as before and L is the maximum bit-size of any algebraic value. For instance, isolating the roots of a polynomial of degree n

*This author is supported by a postdoctoral fellowship from the DAAD, Germany.

†This author is supported by NSF grant #CCR-9002819.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

where each coefficient has at most L bits. Sometimes n is bounded and may be omitted; an example is the GCD of two integers. Most approximation problems (such as the approximate ESP above) naturally belong to the bit framework. In this case, we have an additional input parameter, ϵ. Usually, ϵ is specified by an integer s where $\epsilon = 2^{-s}$.

The main source of trouble in Papadimitriou's paper (and, for that matter, Clarkson's paper) lies in the fact that it is a result in the bit model and yet it lapses into the algebraic framework at several critical moments. This is inadmissible because we know of no sensible combination of the bit framework with the algebraic framework. After all, the NP-hardness of ESP result falls under the bit framework, and the approximation result of Papadimitriou is generally viewed as an approach to circumvent this hardness barrier. As an example of the lapses, note that lengths of polygonal paths are sums of square-roots of rational numbers. These values are no problem to compute in the algebraic framework, but in the bit framework, one must work out the precision to which one need to compute these values. We will see that the precision needed depends on how the numbers are to be used. One cannot take this precision issue lightly as it is precisely (no pun intended) the large precision needed for the sums of square-roots computation that prevents us from placing the ESP problem (even the planar case) into the class NP.

Once one realizes these lapses, it is still not a routine matter to decide the precision to which we must compute these algebraic numbers. We will employ basic techniques of root bounds and algebraic computation (see [9]). One complication is that we will be using approximations of approximations (and so on). The second difficulty is that we need to invoke Brent's results [1] concerning the evaluation of elementary functions to carry out the various approximations. But application of Brent's results requires some arguments, as we will see.

The heart of Papadimitriou's algorithm is a subdivision scheme to break up edges into smaller segments. In our reworking of Papadimitriou's approach, we introduce a new subdivision scheme that has two nice features. If ϵ is halved, we can reuse all the points in the previous subdivision. This can save recomputation in practice (although we do not know if this is an asymptotic improvement). Further, it turns out that our subdivision uses fewer points (this is an asymptotic improvement when ϵ is small enough).

The rest of this abstract is organized as follows: In section 2, we sketch Papadimitriou's algorithm in three parts and indicate where the difficulties come into each part. In section 3, the new subdivision method for edges is described. In section 4, we give some general lemmas on application of Brent's complexity bounds for floating-point number computations. In sections 5 to 8, we sketch the fixes to the original algorithm. We conclude in section 9.

2 Outline of an Approximate Algorithm

Assume we are given a collection of polyhedral obstacles, source and target points s_0, t_0, and an $0 < \epsilon \leq 1/2$. We assume that the obstacles are bounded by triangular facets. There are a total of n edges in the collection of obstacles. Each vertex of obstacles and s_0, t_0 are specified by L-bit integers. Throughout this paper, we use a parameter

$$\epsilon_1 := \frac{\epsilon}{C_0 n},$$

where C_0 is a suitably large constant. For the present discussion, $C_0 \geq 8$ suffices. The algorithm of Papadimitriou can be divided into three parts:

(I) We divide each edge e into a set of *segments* by introducing points p_0, $p_{\pm 1}$, $p_{\pm 2}, \ldots$. We call the p_i's the *break points* and p_0 the *origin* of e. Each segment is of the form $\sigma_i = [p_i, p_{i+1}]$ for $i \in \mathbb{Z}$.

(II) The *visibility graph* $G = (N, A)$ is constructed. The nodes N of the graph comprise the segments from (I) and also s_0, t_0. We can view s_0 and t_0 as degenerate segments. The *visibility edges* A comprise pairs (σ, σ') of nodes that can *see each other*, meaning that there exists $x \in \sigma, x' \in \sigma'$ such that the line segment $[x, x']$ avoids the (interior of) obstacles. We call such a line segment $[x, x']$ a *visibility segment.*

(III) We compute for each visibility edge (σ, σ') a "nominal cost" $C(\sigma, \sigma')$, defined to be the Euclidean distance between the midpoints of σ and σ'. We now apply Dijkstra's shortest path algorithm to the weighted graph $G' = (N, A; s_0, C)$. The shortest path π in G' from s_0 to t_0 is output.

The correctness of the algorithm amounts to the claim that the cost $C(\pi)$ is at most $(1+\epsilon)C(\pi^*)$ where $C(\pi^*)$ denotes the Euclidean length of an actual shortest path π^*. According to the original paper, everything hinges on a clever subdivision scheme for the points $p_{\pm i}$ $(i = 0, 1, \ldots)$ in part (I).

We now give some counter indications:

(I)' The origin p_0 of an edge e is a specially chosen rational point. For $|i| > 0$, $||p_0 - p_i||_2$ is the rational number $\epsilon_1||s_0 - p_0||_2(1+\epsilon_1)^{|i|-1}$. But this means that the coordinates of p_i would in general be irrational. The original paper assumes that p_i can be exactly determined. But in fact, we must determine the precision to which to compute the points. This precision depends on the subsequent use of these points – see (III)' below.

(II)' For any edge e, let *span*(e) denote the line through e. In general, for any set X of points, *span*(X) is the affine span of X. Following Papadimitriou, let us fix two edges e, f and let x and y be real parameters for points in *span*(e) and *span*(f), respectively. Let $e(x)$ and $f(y)$ denote the corresponding points. Now consider a triangular facet T bounding an obstacle. We view T as the intersection of three half-planes contained in *span*(T). Let H be one such half-plane. So its boundary ∂H contains an edge of T. The set of potential visibility segments $[e(x), f(y)]$ such that *span*$(e(x), f(y))$ intersects ∂H is a hyperbola

$$(x-a)(y-b) = c$$

for some a, b, c, as noted by Papadimitriou and Guibas [6]. However, the set H^* of points (x, y) corresponding to potential visibility segments that intersect H is *never* of the form $(x-a)(y-b) \leq c$ or $(x-a)(y-b) \geq c$, contrary to [6]. This is intuitively obvious since the hyperbola is solely determined by ∂H, and does not take into account the actual plane *span*(H). Assuming that we correctly figure out the *configuration obstacle* H^* corresponding to any half-plane H, the corresponding configuration obstacle T^* of T is now given by

$$T^* = H_1^* \cap H_2^* \cap H_3^*$$

where $H_1 \cap H_2 \cap H_3 = T$. Finally, the free-space FP in the (x, y) parameter space is the complement of the union of all the T^*'s. The boundary of FP will consist of axes-parallel line segments and hyperbolic segments. In the plane sweep algorithm proposed to determine FP, the priority of events are now the x-coordinates of intersections of these straight lines and hyperbolas. The original paper improperly assumes that we can compute these priorities exactly.

(III)' The original paper assumes that we can compute the nominal cost function $C(\sigma, \sigma')$ exactly. There are two sources of inaccuracies: we can only compute the midpoints of σ approximately, and then we only compute an approximation to the distance between two approximate midpoints. A more serious matter arises: suppose the algorithm outputs a sequence of segments

$$\sigma_1, \sigma_2, \ldots, \sigma_k$$

that purports to be an approximation to the shortest path. This means that there exists a *zig-zag* path

$$\pi = (s, q_1, r_1, \ldots, q_k, r_k, t)$$

(where $q_i, r_i \in \sigma_i$) that avoids all obstacles. (Note that q_i, r_i can be quite far apart in absolute terms.) We must compute an approximation to this zig-zag path, since obtaining this explicit path is the *raison d'être* of most applications. The approximation must be *guaranteed* to be obstacle-avoiding (and still achieves an ϵ-approximation).

3 A New Subdivision Scheme

In this section, we present a new subdivision scheme for edges. The original scheme would work as well, but our new scheme is motivated by the desire that the subdivision for $\epsilon/2$ is a refinement of the subdivision for ϵ. As in the original, the origin p_0 of an edge e is chosen to be the closest point on e to the source s_0.

We now describe the subdivision of the half-edge e' to one side of the origin (the other half-edge is similarly treated). Let $d = ||s_0 - p_0||_2$ be this closest distance. Choose p_1 on half-edge e' at distance d from p_0. The segment $[p_0, p_1]$ is uniformly subdivided into segments of length $d\epsilon_1/2$ by introducing the points $q_1, q_2, \ldots$. The rest of the half-edge e' is subdivided using the points $p_2, p_3, \ldots$ defined as follows. Let m be the smallest integer satisfying

$$2^m \geq \frac{\log(\ell(e)/d)}{\log(1+\epsilon_1)},$$

where $\ell(e)$ denotes the length of an edge e. Then we define p_i such that

$$||p_0 - p_i||_2 = d\left(\frac{\ell(e)}{d}\right)^{(i-1)/2^m}.$$

Let $D(\sigma, \sigma')$ denote the shortest distance between the two segments σ and σ'.

Lemma 1
(i) The subdivision satisfies the basic inequality

$$\ell(\sigma) \leq \epsilon_1 D(s_0, \sigma)$$

(ii) There are at most $N = O(n(L/\epsilon_1))$ segments in all the edges.
(iii) The subdivision for $\epsilon/2$ is a refinement of the subdivision for ϵ.

Property (i) is similar to the original scheme. Property (ii) improves on the original by an additive factor of $O(n\log(1/\epsilon_1)/\epsilon_1)$. This improvement can be significant because ϵ might be exponentially small in L, especially if we want to guarantee that we approach an actual shortest path. Property (iii) is advantageous to have when we want to use successive refinements of ϵ. Note that Papadimitriou's scheme does not enjoy such a feature. We note two possible disadvantages for our scheme: We use two distinct methods to subdivide e – for the region near the origin, the scheme is uniform and the rest uses an exponential scheme. It seems that both parts are essential to get our properties (ii) and (iii). Also, our segment endpoints (unlike Papadimitriou's) are at irrational distances from the origin.

4 Floating Point Number Computations

Approximate computations introduce a host of detailed calculations which we highlight in this section. Our subdivision scheme requires us to compute values of the form E^F where E, F are numerical expressions. The following captures what is needed:

> (P1) Let s, L_1, L_2 be given natural numbers and E, F be numerical expressions satisfying $2^{-L_1} \leq F \leq 1$ and $-L_2 < \ln E < -2^{-L_2}$. The problem is to compute E^F to precision s, in time polynomial in s, L_1, L_2.

In application, we have $s+L_1+L_2 = O(\log(1/\epsilon_1)+L)$. Note that $\widehat{x}$ approximates x to *precision* (resp., *relative precision*) s means $|\widehat{x}-x| < 2^{-s}$ (resp., $|(\widehat{x}-x)/x| < 2^{-s}$). We resort to Brent's results [1] on computation of elementary functions, and compute E^F using the following obvious straightline program: (i) Compute V_1, V_2 which approximate E, F to precision s_1 and s_2, respectively; (ii) Compute V_3 which approximates $\ln(V_1)$ to precision s_3; (iii) Compute V_4 which approximates $V_2 \cdot V_3$ to precision s_4; finally, (iv) Compute V_5 which approximates $\exp(V_4)$ to precision s_5. We conclude that (with suitable choices of s_i's) V_5 approximates E^F to precision s. We would hope that the cost of these steps is polynomial in s, L_1, L_2. The sequence of approximations is complicated by the fact that application of Brent's result is not straightforward: his bounds for elementary functions assume that each function is defined over an arbitrary but fixed domain $[a,b]$. Under this assumption, one can compute $\exp(x)$ and $\ln(x)$ to precision s in time $O(\mu(s)\log s)$ where

$$\mu(s)$$

denotes the bit-complexity of multiplying s-bit integers. Towards a satisfactory solution of (P1), consider a subproblem:

> (P2) Let s, L be given natural numbers and x a number satisfying $2^{-L} < |x| < 2^L$. Compute $\exp(x)$ to precision s in time polynomial in $s, |x|$.

Floating point numbers. Let us briefly describe the floating point number system we will be assuming. A floating point (f.p.) number α is a pair $\langle e, f\rangle$ of binary integers where e is the *exponent* and f the *fraction*. Let $\langle f\rangle$ denote the number obtained by placing a binary decimal point just in front of the most significant one ('1') in f (if $f = 0$ then define $\langle f\rangle = 0$). Thus $1/2 \leq \langle f\rangle < 1$ for $f \neq 0$. The number *represented by* α is $2^e \cdot \langle f\rangle$. We also write $\alpha \sim 2^e\langle f\rangle$, to emphasize that α is the representation itself, not just the value it represents. The *precision* of α is[1] equal to $1 + \lfloor \lg f\rfloor$

Lemma 2 *Assuming a floating-point representation of numbers, we can solve (P2) in time* $O(\mu(s+|x|)\log(s+|x|)$ *if* $1 \leq |x| \leq 2^L$, *and* $O(\mu(s+\log L)(L+\log(s+\log L)))$ *if* $2^{-L} \leq |x| \leq 1$.

Let us sketch a proof of lemma 2. Let $s' = s + L$ and $\langle e, f\rangle \sim x$. Then

$$\langle 1, f\rangle \sim y := x \cdot 2^{-\lfloor \lg x\rfloor}$$

which lies in the range $[1,2)$. Applying Brent's result, we can compute a value z which approximates $\exp(y)$ to precision s' in time $O(\mu(s')\lg s')$. Now we apply repeated squaring to z for a total of $\lfloor \lg x\rfloor$ times, each time taking care to truncate the result to precision s'. For $|x| < 1$, we do repeated square roots instead of squarings. We then argue that the final result approximates $\exp(x)$ to precision s.

The upshot of a series of arguments similar to the above eventually leads us to:

Lemma 3 *Assume that we can compute E, F to any precision r in time $T_E(r), T_F(r)$, respectively. Then we can solve problem (P1) in time*

$$\begin{aligned} O(&T_F(s+L_0) + T_E(s+L_2) + \\ &\max\{\mu(s+L_2)\log(s+L_2), \\ &\mu(s+\lg(L_0))(L_0+\log(s+\lg(L_0)))\}), \end{aligned}$$

[1] We write 'log' when the actual base is irrelevant. But $\lg = \log_2$ and $\ln = \log_e$.

where we write L_0 for $L_1 + L_2$. If $L_0 = O(s)$, this simplifies to $O(T_E(s) + T_F(s) + \mu(s)s)$.

5 Configuration Obstacles

Fix any two edges e_1, e_2 of obstacles. Write $e_i = [p_i, q_i]$ and let us parameterize the points on $span(e_i)$ such that $xp_i + (1-x)q_i$ is denoted by $e_i(x)$, for $x \in \mathbb{R}$. The set of pairs (x, y) corresponding to a potential visibility segment $[e_1(x), e_2(y)]$ is thus parameterized by the unit square $Q = [0, 1] \times [0, 1]$. Let Q be embedded in $Q' = \mathbb{R}^2$, which denotes the parameter space for $[span(e_1) \times span(e_2)]$. For any obstacle $X \subseteq \mathbb{R}^3$, let X^* denote the set of points $(x, y) \in Q'$ such that $[e_1(x), e_2(y)]$ intersects the relative interior of X. We call X^* the *configuration obstacle* corresponding to X. As in section 2, it is enough to understand the configuration obstacle H^* where H is a half-plane. Assume that ∂H contains the edge $[u, v]$ of a triangular face. Let $\ell(x, y)$ be the line passing through the points $e_1(x), e_2(y)$. The requirement that $\ell(x, y)$ intersects ∂H reduces to the equation $Axy + By + Cx + D = 0$ where $A = \det(v-u \mid p_1-q_1 \mid p_2-q_2)$, $B = \det(v-u \mid q_1 - u \mid p_2 - q_2)$, $C = \det(v - u \mid p_1 - q_1 \mid q_2 - u)$ and $D = \det(v - u \mid q_1 - u \mid q_2 - u)$. If $A \neq 0$, we may rewrite the equation in a standard form for hyperbolas: $(Ax + B)(Ay + C) = -AD + BC$.

Degenerate position. We say the two edges e_1, e_2 are in *degenerate position* with respect to H if $A = 0$ or $(A \neq 0) \wedge (AD - BC = 0)$. Let us analyze two edges in degenerate position. If $A = 0$ then it can be shown that either (i) the points $u, v, u+p_1-q_1, u+p_2-q_2$ are on the same plane, or (ii) the two edges e_1, e_2 are in two parallel planes that do not intersect with the line ∂H. Now consider the case $A \neq 0$. Using the relation (see [5])

$$\begin{aligned} \det(a \mid b \mid d) \det(a \mid c \mid e) = \\ \det(a \mid b \mid c) \det(a \mid d \mid e) \\ - \det(a \mid b \mid e) \det(a \mid d \mid c). \end{aligned}$$

we may deduce

Lemma 4

(i) $$AD - BC = \det \begin{pmatrix} u & v & u + p_1 - q_1 & q_1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \cdot \det \begin{pmatrix} u & v & u + p_2 - q_2 & q_2 \\ 1 & 1 & 1 & 1 \end{pmatrix}.$$

(ii) If $AD - BC = 0$ then one of the lines $span(e_1), span(e_2)$ either is parallel with ∂H, or intersects ∂H.

Suppose e_1, e_2 is not in degenerate position with respect to H. It can be shown that the sign of $AD - BC$ depends on the direction of vector $p_i - q_i$ with respect to H and on whether $span(e_i)$ intersects H, for $i = 1, 2$.

Configuration Obstacles. We are ready to describe $H^* \subseteq Q'$. Let us define two points $(x_0, y_0), (x_1, y_1) \in Q'$ as follows: $span(H)$ intersects $span(e_1)$ at $e_1(x_0)$ and intersects $span(e_2)$ at $e_2(y_0)$. Let P_i be the plane containing ∂H and parallel to e_i. Then P_1 intersects $span(e_2)$ at $e_2(y_1)$ and P_2 intersects $span(e_1)$ at $e_1(x_1)$. We may assume that these intersection points are well-defined. Furthermore, imagine $span(H)$ to be the horizontal plane $\{Z = 0\}$ in physical (X, Y, Z)-space, and that $p_i - q_i$ have a positive Z-component. The vertical and horizontal lines through (x_0, y_0) divide Q' into four quadrants. By symmetry, we may assume that e_1, e_2 are parameterized to increase in the positive Z direction. Then the first and third quadrants are necessarily disjoint from H^*. Let us consider one possibility (the other is similarly treated), where the line ∂H separates $e_1(x_0)$ and $e_2(y_0)$ on the plane $span(H)$. We may assume (x_1, y_1) lies in the first quadrant. See figure 1. The asymptotes of the hyperbola in this case are the vertical and horizontal lines through (x_1, y_1), and (x_0, y_0) lies on the hyperbola. Suppose the hyperbola lies to the northwest and southeast of (x_1, y_1). Then H^* is shown as shaded in figure 1.

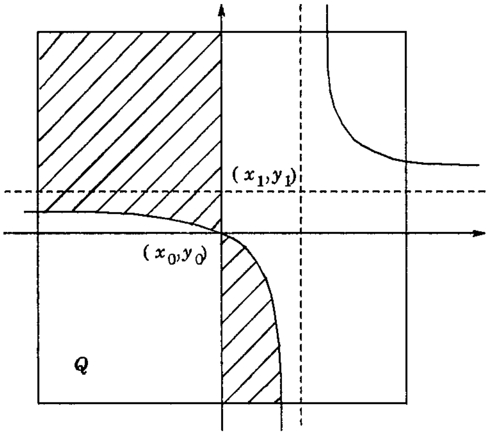


Figure 1. Configuration obstacle for a half-space.

6 Construction of the visibility graph

In this section, we shall (like Papadimitrou) count the number of algebraic operations, to give some idea of the global complexity. To avoid confusion with the bit-complexity accounting in the next section, we call each algebraic operation a "step".

Again focus on a pair of fixed edges e_1, e_2. The break points (these are the p_i's and q_i's in section 3) on e_1 and e_2 define a grid on Q. For segments $\sigma_i \subseteq e_i$ $(i = 1, 2)$, there is a corresponding *grid rectangle* $R = R(\sigma_1, \sigma_2) \subseteq Q$. We say that R is *covered* iff

$$R(\sigma_1, \sigma_2) \setminus K^* = \emptyset. \qquad (1)$$

Clearly (σ_1, σ_2) is an edge in the visibility graph G iff $R(\sigma_1, \sigma_2)$ is not covered.

Our goal is to decide for every grid rectangle R whether or not it is covered; if it is uncovered, we will further determine a rational sample point p such that a ball of some small radius δ centered at p is contained in $R \setminus K^*$. Note that this determination will allow us to compute a *guaranteed* obstacle-avoiding path.

Let

$$M = O(L/\epsilon_1)$$

be a bound on the maximum number of break points on any edge. Then there are at most M^2 grid rectangles in Q. We proceed slightly differently than Papadimitriou.

First we compute the arrangement of the collection of boundary arcs corresponding to all the configuration obstacles T^*. This can be computed in $O(n\lambda_4(n))$ where $\lambda_4(n) = O(n \cdot 2^{\alpha(n)})$ and $\alpha(n)$ is the inverse of an Ackermann-like function. This follows from the fact that any two boundary arcs can intersect at most twice, and from the incremental algorithm in [4]. We next convert this arrangement into an explicit representation of the boundary of the configuration obstacle K^*, say, in $O(n^2 \log n)$ steps. Finally we perform an interesting kind of line sweep over this representation of K^* in Q. Our goal is to decide for each grid rectangle whether it is covered by K^*. There are three kinds of events for the sweepline:

(i) Vertices on the boundary of K^*.
(ii) The break points $p_0, p_{\pm 1}, p_{\pm 2}, \ldots$ on e_1.
(iii) Intersections between a horizontal grid line and a boundary arc of K^*. Unlike the first two types, this is dynamically inserted into the priority queue.

The state of the sweepline is represented by a balanced binary tree T that contains a leaf for each of the horizontal grid lines and for each of the boundary arcs that intersect the sweepline. Whenever a grid line and a boundary arc become adjacent in T, we check if they will generate an event of type (iii). Whenever we come to an event of type (ii), we have just completely swept a column of at most M grid rectangles and will begin to sweep a new column of at most M grid rectangles. We use $O(M)$ steps to do the processing of these grid rectangles. There are $O(n^2)$ events of type (i) and $O(nM)$ events of type (iii). They can each be processed in $O(\log(n + M))$ steps. The events of type (ii) take $O(M)$ steps each, and $O(M^2)$ altogether.

7 Bit Complexity

We must now determine the bit complexity of the algorithm in the preceding section. In particular, this calls for determining the precision to carry out the various tasks. We need this, for example, to make sure that we can do line sweep correctly and to be able to determine sample points in each grid rectangle R that is not covered. Towards this end, we shall employ a variety of classical techniques in algebraic computing [9]. The upshot of the calculations (sketched below) is that

$$W = O(\log(1/\epsilon_1) + L)$$

bits of precision suffices in the computation of points. So comparisons between numbers in the algorithm (as in priority queue operations) take $O(W)$ time.

For a given pair of edges, the following bit complexity is readily determined: The break points $p_{\pm i}, q_{\pm j}$ will be approximated to $O(W)$ bits, and by lemma 3, these can be computed altogether in $O(M\mu(W)W)$ time. We can compute the arrangement in time $O(n\lambda_4(n)\mu(W))$. To compute the boundary of K^* takes $O(n^2 \log n \cdot W)$. The sweepline operations take $O(W)$ per queue operation and $O(\mu(W))$ to compute events of type (iii). Thus all the operations of type (i) cost $O(n^2 \log(n + M)W)$; type (ii) cost $O(M^2 W)$ and type (iii) cost $O(nM \log(n + M)\mu(W))$. Summarizing, the bit complexity of the algorithm in the previous section, applied to all pairs of edges, is

$$O(n^3\lambda_4(n)\mu(W) + n^3 M \log M\mu(W) \\ +(nM)^2 W + nM\mu(W)W).$$

We now go into some details. For simplicity, we will ignore additive constants in the size of integer numbers.

There are 3 types of event points in the line-sweep algorithm: the x-coordinates of intersection points between boundary curves, of asymptotes of hyperbolic curves, and of parameters for break points. In the following, we talk about "separating a set of points" –

this means we compute the x- and y-coordinates of these points to sufficient precision so that the relative x-order and y-order of any two points in the set are correctly determined. This is needed for the line-sweep algorithm to work correctly.

The precision to separate two break points depends on the minimum distance between two consecutive break points, i.e., the shortest length of line segments. We have:

Lemma 5 *A precision of* $O(\log(1/\epsilon_1)+L)$ *suffices to separate two break points.*

Now consider the intersection points of two hyperbolic curves. From the preceding development, if $Axy+By+Cx+D=0$ is the equation of a hyperbola that partially bounds a half-space configuration obstacle H^*, then A,B,C,D are integers of size $3L$.

Let α be the x-coordinate of an intersection point. The maximum bit size for α is the sum of (i) bit size of the integral part of the maximum α and (ii) the precision needed to separate any two intersection points or precision needed to separate any intersection point from 0. Let $P_1(x,y) = A_1xy + B_1y + C_1x + D_1, P_2(x,y) = A_2xy + B_2y + C_2x + D_2$ be hyperbolic functions with integer coefficients. Then the x-coordinates of intersection points between $P_1(x,y)$ and $P_2(x,y)$ are roots of the resultant $\text{res}_y(P_1,P_2)$, where

$$\text{res}_y(P_1(x,y),P_2(x,y)) = \det\begin{vmatrix} A_1x+B_1 & C_1x+D_1 \\ A_2x+B_2 & C_2x+D_2 \end{vmatrix}. \quad (2)$$

The resultant is an integer polynomial with coefficients of length $6L$. By Cauchy's bound on roots of polynomials [9], $6L$ bits suffice to represent the integral part of any root, and $6L$ bits suffice to separate the roots from 0.

Next let us calculate a lower bound on the gap between two intersection points, assuming they are different. Let α be the root of the resultant $\text{res}_y(P_1(x,y),P_2(x,y))$ and β be the root of the resultant $\text{res}_y(P_3(x,y),P_4(x,y))$ where P_1,P_2,P_3,P_4 are hyperbolic functions that are integer polynomials. Applying the generalized Hadamard's bound [9] to the two resultants, we show that $24L$ bits are needed to separate α and β. Thus we need $6L, 24L$ bits for the integral, fractional part of the intersection points between two hyperbolic functions, respectively. This yields:

Lemma 6 *A precision of* $30L$ *bits suffices to separate any two intersection points of hyperbolic curves.*

Next we determine the precision to separate the intersection points and the approximate break points in the parameter space Q'. Let $P(x) = Ax^2+Bx+C$ be the polynomial in (2). The roots of $P(x)$ contain the x-coordinates of intersection points between two hyperbolic curves. The coefficients of P are integers of length $6L$. The approximate position of break points is represented by a rational number D/E where $E=2^W$ and D is an integer such that $|D|\le E$. Therefore it is the root of $Q(x)=Ex-D$. The generalized Hadamard's bound again shows that $2W+12L$ bits suffice to separate the roots of $P(x)$ and $Q(x)$. Summing up the precisions that suffice for integral and fractional parts of points, we conclude:

Lemma 7 *A precision of* $2W+18L = O(\log(1/\epsilon_1)+L)$ *suffices to separate intersection points of hyperbolic curves from break points.*

In the same way, we can show that other types of pairs of events are separated with this precision.

Theorem 8 *It suffices to compute the event points to precision* $O(\log(1/\epsilon_1)+L)$.

8 Coda

Now that we have a visibility graph $G=(N,A)$, we wish to compute the nominal cost of each visibility edge in A. Unlike the original algorithm, we do not pick the distance between midpoints of segments for computing nominal costs. Instead, we pick *any* rational point in each segment instead of the midpoint. If we run Djikstra's algorithm on G, assuming that we know the *exact* nominal cost of each visibility edge, then the original calculations shows that the distance of the "approximate shortest path" π is at most $1+\epsilon$ times the length of the actual shortest path π^*.

But, of course, our calculation only uses approximations to the nominal cost. The approximations arise because we must take square roots. Because we choose rational points, it turns out that we only need to compute square-roots of rational numbers x where $2^{-4L-6}\le x\le 2^{2L+4}$ to some precision s. It turns out that

$$s=\log(4n/\epsilon)$$

suffices, and it takes $O(\mu(W))$ time to compute the square-root to this precision.

After this, we may invoke Dijkstra's algorithm and compute the approximate shortest path π. Note that all intermediate numbers in Dijkstra's are sums of at most n approximate distances, and hence they have sizes at most $O(\ln(n/\epsilon) + L)$. Each operation in Dijkstra's algorithm involving these numbers takes time $O(\ln(n/\epsilon) + L)$ since they are either addition of two numbers or comparisons. Thus the bit complexity of Dijkstra's algorithm is

$$O(N^2 \mu(W))$$

where $N := nM$ is an upper bound on the number of segments in all the edges.

One final point: suppose Dijkstra's algorithm yields a sequence of segments $(s_0, \sigma_1, \sigma_2, \ldots, \sigma_k, t_0)$. For each consecutive pair (σ_i, σ_{i+1}) of segments, we can compute a *sample point* $(x_i, y_i) \in \sigma_i \times \sigma_{i+1}$ such that the segment $[x_i, y_i]$ is a visibility segment (section 2(III)). In fact, in the parameter space, we can ensure that there is a ball B_i of positive radius centered around (x_i, y_i) such that every $(x', y') \in B_i$ represents a visibility segment. Using the sample points, we finally obtain an explicit zig-zag path (section 2(III)') that is obstacle avoiding. In conclusion:

Theorem 9 *There is an approximate algorithm for ESP in 3-space that computes an explicit obstacle-avoiding path with relative error $\epsilon > 0$ with bit-complexity*

$$O((n^3 \lambda_4(n) + n^3 M \log M + (nM)^2) \cdot \mu(W))$$

where $W = O(\ln(n/\epsilon) + L)$ and $M = O(nL/\epsilon)$.

9 Conclusion

1. Papadimitriou's paper on approximate Euclidean shortest paths has become recognized as a key result in this area. In this paper, we rectified several gaps in his paper, and presented for the first time, a true bit-complexity bound for an ϵ-approximation algorithm for ESP.

2. It should be clear from our development that there are many tedious and non-obvious details when we do such approximations. The result is an interesting mix of combinatorial, numerical and algebraic complexity. Many of these tools are, in fact, those that are necessary in order to make "exact computation" a practical reality [8].

3. In our solution, we tried to extract some general lemmas about approximate computing with floating point numbers, to facilitate future applications of such an approach. Note that one could avoid Brent's result by approximating Taylor's expansions directly, but this is expected to yield an inferior bit complexity.

4. We may generally call this the method of *generalized grids*. Of course, grids are a familiar practical technique in all of computational sciences. From a complexity theoretic viewpoint, such methods have been shunned in the past as trivial or uninteresting. This need not be so, as Papadimitriou's work has demonstrated. In fact, grid methods may be the most practical recourse for solving some intractable problems. It would be interesting to derive some general theorems about these approaches.

References

[1] Richard P. Brent. Fast multiple-precision evaluation of elementary functions. *Journal of the ACM*, 23:242–251, 1976.

[2] J. Canny and J. H. Reif. New lower bound techniques for robot motion planning problems. *IEEE Foundations of Computer Science*, 28:49–60, 1987.

[3] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. 19th Annu. ACM Sympos. Theory Comput.*, pages 56–65, 1987.

[4] H. Edelsbrunner, L. Guibas, J. Pach, R. Pollack, R. Seidel, and M. Sharir. Arrangements of curves in the plane: topology, combinatorics, and algorithms. *Theoret. Comput. Sci.*, 92:319–336, 1992.

[5] H. Lass. *Vector and Tensor Analysis.* McGraw-Hill, 1950.

[6] C. H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Inform. Process. Lett.*, 20:259–263, 1985.

[7] Subhash Suri and John Hershberger. Efficient computation of euclidean shortest paths in the plane. *IEEE Foundations of Computer Science*, pages 508–517, 1993. An improved $O(n \log n)$ bound was recently announced.

[8] Chee Yap. Towards exact geometric computation. In *Fifth Canadian Conference on Computational Geometry*, pages 405–419, Waterloo, Canada, August 5–9 1993. Invited Lecture.

[9] Chee Yap. *Fundamental Problems in Algorithmic Algebra.* Princeton University Press, to appear. Available on request from author (and via anonymous ftp).

The Realization Problem for Euclidean Minimum Spanning Trees is NP-hard

Peter Eades*
Department of Computer Science
University of Newcastle
Newcastle Australia

Sue Whitesides†
School of Computer Science
McGill University
Montreal Canada

Abstract

We show that deciding whether a tree can be drawn in the plane so that it is the Euclidean minimum spanning tree of the locations of its vertices is NP-hard.

Keywords: *tree, algorithm, graph drawing, graph layout, NP-hard, spanning tree.*

1 Introduction

This paper investigates a fundamental geometric realization problem, the *realization problem for Euclidean minimum spanning trees*: Given a tree $T = (V, E)$, find a location $D(u)$ for each vertex $u \in V$ so that T is a minimum spanning tree of $\{D(u) : u \in V\}$, or determine that no such locations exist.

This problem belongs to the growing body of knowledge and research on the geometric realizability of combinatorial objects and the related problem of finding combinatorial characterizations of geometric objects. The Delaunay triangulation realizability problem [4], [3] provides another such example. It asks whether, for a given triangulation specified combinatorially in terms of vertices, edges, faces and incidence relations, there is a way to place the vertices in the plane so that their Delaunay triangulation is combinatorially equivalent to the given triangulation.

The study of graph drawing and layout also prompts interest in Euclidean minimum spanning tree realizability. Several graph drawing algorithms try to make a layout which is as "small" as possible in some sense. Some aim for minimum area, some for minimum width, some for minimum total edge length [12, 10]. We have observed that in a number of experimental results (see [5]), algorithms for drawing trees often (but not always) produce tree layouts that are minimum spanning trees, or nearly minimum spanning trees, of the vertex locations.

In a paper that studies graph theoretic changes in minimum spanning trees of moving point sets, Monma and Suri [9] note that every tree with maximum degree at most five can be drawn as a minimum spanning tree, and that no tree with a vertex of degree larger than 6 can be drawn as a minimum spanning tree.

Bose, Lenhart and Liotta [1] show that for several classes of proximity graphs (for example, relative neighborhood graphs and Gabriel graphs), the problem of determining whether a tree can be realized as a proximity graph can be solved in polynomial time.

This paper presents the first negative result in this area: the realization problem for Euclidean minimum spanning trees is NP-hard.

2 Preliminaries

Given a set S of points in the plane, a *spanning tree* of S is a tree whose vertex set is S. Edges of the tree can be regarded either as pairs of points in S or as open line segments between points in S. A spanning tree $T = (S, E)$ of S is a *Euclidean minimum spanning tree* of S if

$$\sum_{uv \in E} dist(u, v)$$

is minimized over all spanning trees of S.

[*] eades@cs.newcastle.edu.au Dept. of C.S., University Dr., Callaghan NSW 2308 AUSTRALIA

[†] sue@cs.mcgill.ca School of C.S., 3480 University St. #318, Montreal, Quebec H3A 2A7 CANADA

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

A *drawing* D of a tree $T = (V, E)$ is a pair of 1-1 functions

$$D_V : V \rightarrow R^2,\ D_E : E \rightarrow L,$$

where L is the set of open line segments in R^2, such that the endpoints of $D_E(u, v)$ are $D(u)$ and $D(v)$. The images of the vertices and edges of T under D form a tree which is isomorphic to T.

When discussing a tree $T = (V, E)$ and some drawing D of T, we often use the same notation for a vertex and its position as given by D_V, and the same notation for an edge and the corresponding line segment. For instance, if $u, v, a, b \in V$ and $uv, ab \in E$ then expressions such as "the length of the edge uv" and "edge ab intersects edge uv" refer to the drawing D of T. Also, a "vertex" or "edge" of D refers to the image under D_V or D_E of a vertex or an edge of T.

Definition 1 *Suppose that D is a drawing of $T = (V, E)$, and let $S = \{D_V(v) : v \in V\}$. If T is isomorphic to a minimum spanning tree of S, then we say that D* realizes *T.*

Thus "T is realizable" means that there is a 1-1 function D_V that maps each tree vertex v to a point $D_V(v)$ in the plane, such that T is isomorphic to at least one of the Euclidean minimum spanning trees of the point set $D_V(V)$. Hence our main result can be described as follows.

Minimum Spanning Tree Realizability (MSTR)
Instance: A tree T.
Question: Is T realizable?

Theorem 1 *MSTR is NP-hard.*

Specifically, we show that the problem of determining whether trees with maximum degree 6 are realizable is NP-hard.

We say a tree T has a *unique realization D*, or that D *uniquely realizes* T, if D realizes T and for every other realization D' of T, there is a congruence mapping $D_V(V)$ to $D'_V(V)$ and $D_E(E)$ to $D'_E(E)$ after a suitable change of scale.

We now give a brief overview of the proof of the main result. We prove Theorem 1 by a reduction from Not-All-Equal-3SAT (see Section 5). This is done by designing a family of trees we call *(m, n)-logipedes.* These trees are defined *purely combinatorially* in terms of incidences of vertices and edges. We show that logipedes are uniquely realizable. Figure 1 shows the unique realization of a $(3, 4)$-logipede.

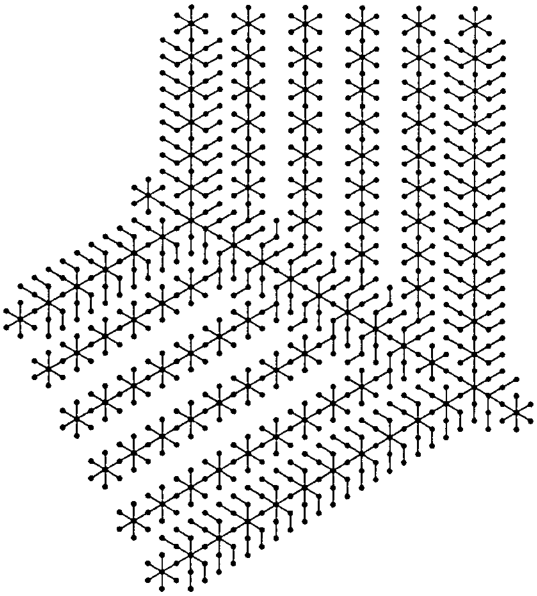


Figure 1: A (3,4)-logipede

The realization shown in the Figure places the tree on a triangular grid. We prove that this phenomenon holds in general. Any realization of a logipede as a Euclidean minimum spanning tree draws the logipede on a triangular grid. Here we emphasize strongly that the combinatorial structure of the tree forces the grid to appear implicitly in the realization. No initial requirement is made that the tree must be embedded on a grid, or that its edge lengths must be uniform.

To transform an instance of Not-All-Equal-3SAT with m clauses and n variables to an instance of MSTR, we construct an (m, n)-logipede and use it to provide a framework for a pair of (clause, boolean variable) and (clause, complement variable) incidence matrices. We regard the realization of an (m, n)-logipede as consisting of a "spine" whose length is proportional to n, to which pairs of "legs" are attached whose lengths depend on m and the place of attachment along the spine. Each pair of legs represents a boolean variable and its complement, and the legs in the pair act like columns in the two incidence matrices. For technical reasons, the legs grow in length, and so the rows of the incidence matrices cannot be visualized as parallel to the spine, but rather as forming an angle with it. Details are given in Section 5.

The occurence or not of a literal in a clause is indicated by modifying the leg corresponding to the literal at the position in the leg that corresponds to the clause.

To define logipedes and to prove that they are uniquely realizable, we construct a sequence of uniquely realizable "gadgets". The next section describes the gadgets. Section 4 indicates how to prove they are uniquely realizable. Section 5 proves Theorem 1.

3 The Gadgets

Since our proof that a particular gadget is uniquely realizable generally depends on knowing that its predecessors are uniquely realizable, the sequence of gadgets given here can be regarded as a route through the proof of logipede unique realizability.

It cannot be over-emphasized that the gadgets are purely combinatorial objects, sometimes called free or abstract trees. To save space, however, we convey the definition of most gadgets by giving a realization rather than by writing out a combinatorial definition.

Definition 2 *A tree consisting of a degree 6 vertex together with its six neighbors is a* star. *The degree 6 vertex is called the* center *of the star, and the other vertices are called the* points *of the star.*

A star is illustrated in two ways in Figure 2: combinatorially, and realized as a minimum spanning tree.

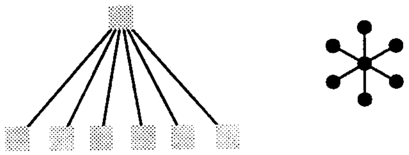


Figure 2: A star and its realization

Chains of stars have limited but multiple possibilities for realizations (Figure 3). Chains of stars are subtrees of many of our gadgets, but they must be modified to achieve unique realizability.

Definition 3 *The tree in Figure 4 is a* base. *The star with center s in the Figure is called the* center star *of the base, and vertex s is called the* center *of the base. The two remaining stars are called* leg stars *of the base.*

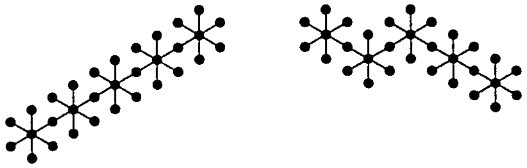


Figure 3: Two realizations of a star chain

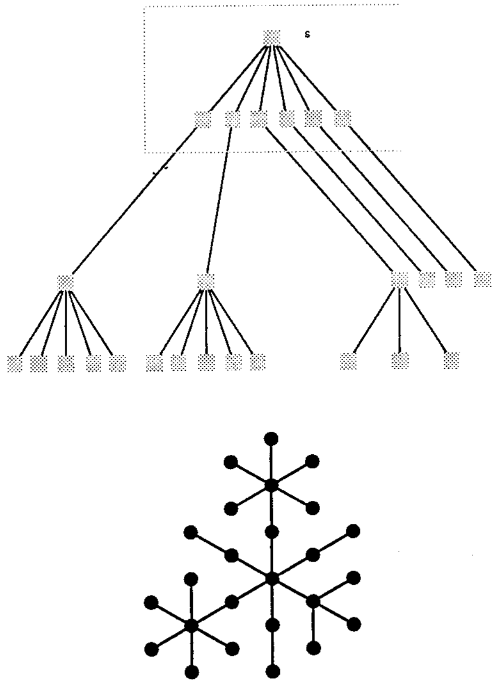


Figure 4: A base and its realization

Definition 4 *The tree in Figure 5 is a* spine base. *The new star is called the 1st or* leading *spine star, denoted S_1.*

Definition 5 *The tree in Figure 6 is a* short spine *in realized form. The two new stars adjoined to the 1st leg stars are called 2nd leg stars. The third new star is called the* cap *star.*

Definition 6 *Figure 7 illustrates a* spine of length k *for $k = 7$. (A spine of length 1 is just a short spine.)*

Definition 7 *Figure 8 gives a realization of an (m, k)-legged spine for $m = 6$ and $k = 7$. We say the spine has*

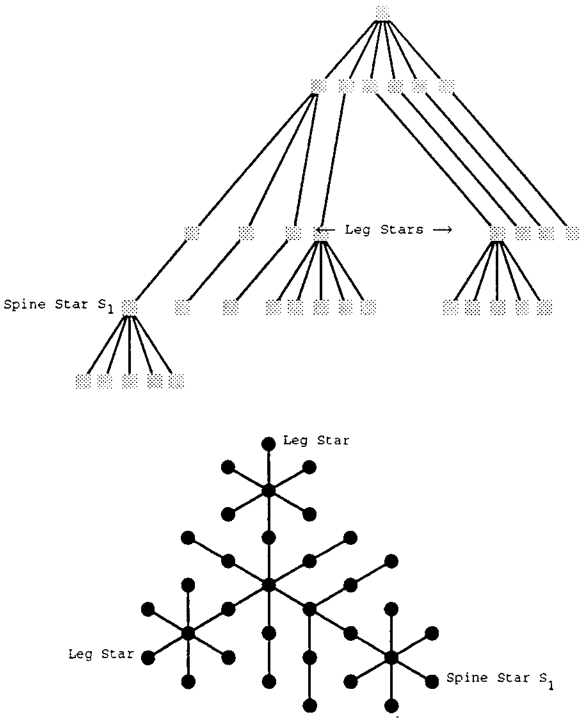


Figure 5: A spine base and its realization

length k and the legs have length m because the number of spine stars is k, and the number of leg stars in each leg is m. (A $(2,k)$-legged spine is just a spine of length k.)

Definition 8 *Figure 9 gives a realization of an (m_1, m_2, k)-*skeleton *for $m_1 = 6$, $m_2 = 11$ and $k = 8$. It is created by joining an $(m_2, 1)$-legged spine to an $(m_1, k-1)$-legged spine as shown. We say it has k spine stars, or length k for short, and front legs of length m_1 and rear legs of length m_2.*

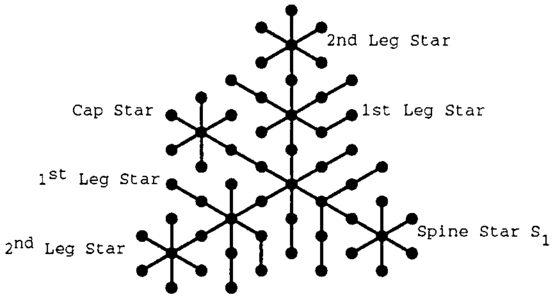


Figure 6: A realization of a short spine

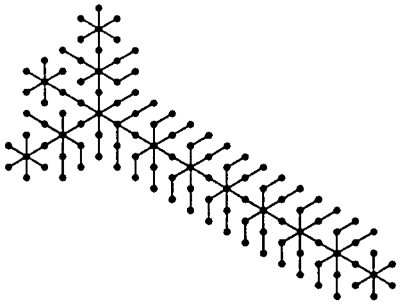


Figure 7: Realization of a spine of length 7

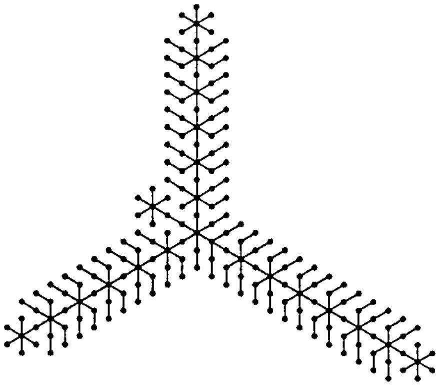


Figure 8: A (6,7)-legged spine

NOTATION. Note that, given an (m_1, m_2, k)-skeleton, the $(m_1, k-1)$-legged spine and the $(m_2, 1)$-legged spine used to create it are uniquely identifiable. The spine stars in the $(m_1, k-1)$-legged spine retain their names in the skeleton, and are called the 1*st*, 2*nd*, ..., $k-1$*st* spine stars (denoted S_1, S_2, ..., S_{k-1}) of the skeleton. The cap star of the $(m_2, 1)$-legged spine becomes the k*th* spine star S_k of the skeleton.

Definition 9 *Figure 1 of the previous section illustrates an* (m,n)-logipede *for $m = 3$ and $n = 4$. An (m,n)-logipede is created from a $(2m, (2m+n+1), 2n)$-skeleton as shown in the figure. Here the m and n refer to the number m of clauses and n of variables in a Not-all-equal-3SAT instance that the logipede will encode, and the numbers of stars in the legs and the spine are functions of these parameters.*

Definition 10 *A* pre-(m,n)-logipede *is the tree obtained by following the construction for an (m,n)-logipede, but adjoining star chains of length 1 instead of star chains whose lengths depend on m and the index of the spine star.*

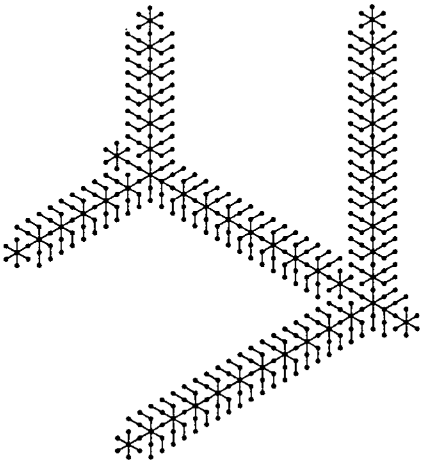


Figure 9: A (6,11,8)-skeleton

4 Uniqueness Results

This section outlines the proof that (m,n)-logipedes are uniquely realizable. Most proofs of lemmas are straightforward and have been omitted. Some are left for flavor.

Lemma 1 *Suppose that S is a finite set of points in the plane, T is a minimum spanning tree of S, and $S' \subseteq S$. If the subgraph T' of T induced by S' is connected, then T' is a minimum spanning tree of S'.*

The above Lemma is used as follows. Suppose that D' is a unique realization of a tree T', and that a tree T contains T'. Then in every realization D of T, T' must be drawn congruent (after scaling) to D'.

Lemma 2 *If segments ab and bc are edges in a Euclidean minimum spanning tree, then $\angle abc$ is the largest angle in $\triangle abc$.*

Lemma 3 *In every realization of a star, all edges have the same length, and consecutive edges in the cyclic ordering of edges around the center form 60° angles.*

Lemma 4 *Let D be a realization of tree T. Suppose that*

1. *D contains three vertices a, b, c such that segments ab and bc are of length l and form an angle of 120°. (These segments are not necessarily edges of D).*
2. *Vertex b is a point of a star with center s, where the star is disjoint from a and c.*
3. *In D, both the path between b and c and the path between a and b contain an edge of length l.*

Then the edge sb has length l and forms an angle of 120° with each of the segments ba and bc.

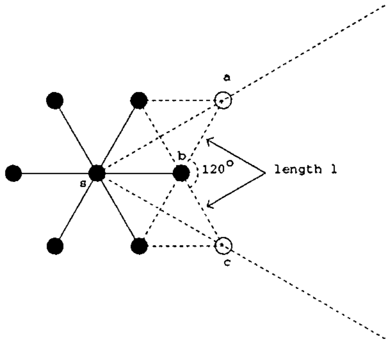


Figure 10: Diagram for 120-degree Lemma

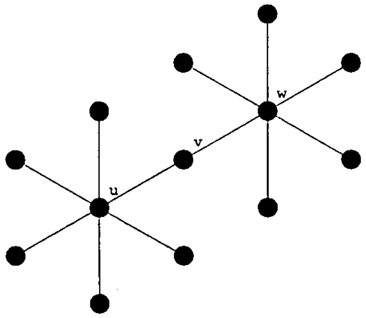


Figure 11: Realization of a double star

Corollary 1 *Suppose that T is a star chain of length 2. Then T is uniquely realizable as in Figure 11.*

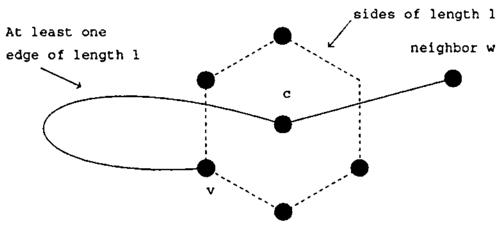


Figure 12: Diagram for 5-hex Lemma

Lemma 5 *Suppose that in some realization D of tree T, a vertex c of T is located at the center of a regular* *hexagon of side length l, at least five of whose vertices are occupied by vertices of T, which need not be adjacent to c or to each other in T. Suppose further that for each vertex v of T located at a vertex of the hexagon, there is a path in D connecting v to c that contains at least one edge of length l (Figure 12). Then each neighbor w of c must be located at one of the 6 vertices of the hexagon.*

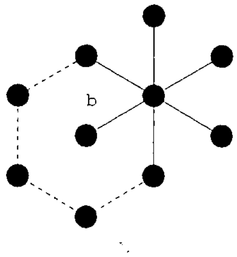


Figure 13: Diagram for 3-claw Lemma

Lemma 6 *Let T be a tree formed be adjoining three new leaves to one of the points b of a star. Then in any realization D of T, b is located at the center of a regular hexagon whose vertices are occupied by the center of the star, two points of the star, and the three new leaves adjoined to point b. (See Figure 13.)*

A vertex of a realization D of a tree T is called a *hexagon center* if it is the center of a regular hexagon of side equal to the minimum edge length in D, such that at least 5 of the vertices of the hexagon are occupied by vertices of D.

Lemma 7 *Suppose that*
1. D is a realization of a tree T in which a vertex c is located at the center of a regular hexagon with side length l; denote the vertices of the hexagon by $h_1, ...h_6$ in cyclic order.
2. Vertices h_1, h_2 and h_3 of the hexagon are occupied by vertices v_1, v_2 and v_3 of D, where for each v_i, $1 \leq i \leq 3$, there is a path in D connecting v_i to c and containing an edge of length l.
3. Vertex c is adjacent in T to a vertex s of degree 6, $s \neq v_1, v_2$ or v_3.
Then:
- Vertex s is located at vertex h_5 of the hexagon, and the edges between s and its neighbors have length l, and
- The two neighbors of s that precede and follow c in the cyclic ordering of the neighbors of s occupy vertices h_4 and h_6 of the hexagon.

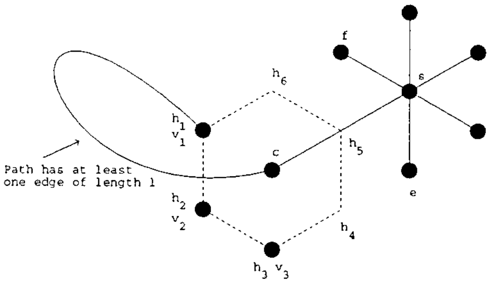


Figure 14: Diagram for 3-hex Lemma.

Proof: In D, let e and f denote the neighbors of s that precede and follow c in the cyclic ordering of vertices adjacent to s, as in Figure 14. The fact that D is a minimum spanning tree places constraints on the locations of e and f. By Lemma 3, the segments ce and cf must form an angle of 120°.

The only solution to these constraints is to place e and f at vertices h_4 and h_6 of the hexagon, which by Lemma 3 determines the placement of s and the set of locations occupied by its neighbors. □

With the machinery of the preceding technical lemmas, it is possible to prove in order the unique realizability of the following sequence of gadgets (we omit the proofs; see the figures of the previous sections for the realizations): spine bases, short spines, (m, k)-legged spines, (m_1, m_2, k)-skeletons, pre-(m, n)-logipedes, and (m, n)-logipedes.

5 The Transformation

This section uses the uniqueness of the logipede realization to transform the NP-complete problem Not-All-Equal-3SAT [6] to MSTR.

Not-All-Equal-3-Sat (NAE3S)
Instance: A set C of m clauses $c_1, c_2, ..., c_m$, each containing 3 literals from a set X of n boolean variables $X_1, X_2, ..., X_n$ and their complements.
Question: Can consistent truth values be assigned to the literals so that each clause contains at least one true literal and at least one false literal?

Without loss of generality, we assume that no clause in C contains both a variable X_j and its complement X_j'.

Our transformation follows the paradigm used by Bhatt and Cosmodakis [11], Idicula [7], Gregori [8] and Brandenburg [2]. We modify an (m,n)-logipede so that the modified logipede can be realized (although not necessarily uniquely) if and only if the NAE3S instance being transformed is a yes instance.

Consider two mxn incidence matrices defined as follows. Row i of each matrix represents clause c_i; in one matrix, column j represents variable X_j, and in the other matrix, column j represents the complement X'_j of variable X_j. In the (clause, variable) matrix, entry (i,j) is 1 if clause i contains variable X_j; otherwise, the entry is 0. The entries of the (clause, complementary-variable) matrix are defined similarly.

An (m,n)-logipede represents these two matrices as follows. Let l_j and l'_j denote the leg chains associated with the jth odd-indexed spine star S_{2j-1}, $1 \leq j \leq n$. These legs represent variable X_j and its complement X'_j, respectively. For geometric reasons, it is convenient to make the first $j-1$ stars closest to the spine in legs l_j and l'_j serve as padding that does not represent any positions of the incidence matrices. Beyond the padding, every other star in a leg chain represents an incidence matrix position. In particular, position (i,j) in the (clause, variable) matrix is represented by a star in leg l_j, namely by star $(j-1)+(2i-1)$, where the leg star closest to the spine is regarded as the first star in the leg.

Lemma 8 *There is a polynomial time transformation from NAE3S to MSTR.*

Proof: Given an NAE3S instance with m clauses and n variables, construct an (m,n)-logipede T, and then add to it as follows.

1. If variable X_j appears in clause c_i (so by assumption X'_j does not appear in c_i), select two degree 1 points of star $(j-1)+(2i-1)$ in leg l_j for modification. Adjoin two new leaves to one of the selected points, and one new leaf to the other selected point. Do not modify leg l'_j.
2. Similarly, if X'_j, but not X_j, appears in c_i, then adjoin new leaves as above to two points of star $(j-1)+(2i-1)$ in leg l'_j but do nothing to leg l_j.
3. If neither X_j nor X'_j appears in c_i, then adjoin new leaves as above both in l_j and in l'_j.

Clearly this transformation can be done in polynomial time in the length of the NAE3S instance. The set of three new leaves adjoined to a star that is modified is called a *striker.* Figure 15 gives one possible realization of the logipede with strikers corresponding to the yes instance $c_1 = X'_1$ OR X'_2 OR X'_3, $c_2 = X'_1$ OR X'_2 OR X'_4, and $c_3 = X'_2$ OR X'_3 OR X_4.

Suppose that the NAE3S instance is a yes instance. The underlying unlabelled (m,n)-logipede has a unique realization. However, the labelled leg l_j could be drawn on either side of the spine, with leg l'_j drawn on the opposite side. Consider a consistent truth assignment to variables such that each clause contains at least one false literal and at least one true literal. A realization of the modified logipede, with strikers added, can be obtained from such a truth assignment by placing all legs whose corresponding literals are true on the same side of the spine, and placing all remaining legs on the opposite side of the spine. The details are straightforward.

To show that only yes instances of NAE3S are transformed to yes instances of MSTR, arbitrarily choose a true side and a false side of the spine in the minimum spanning tree realization. Set the literals corresponding to legs appearing on the true side to true and the remaining literals to false. In the example shown in Figure 15, the assignment of truth values read off from the realization would be $X_1 = X_3 = X_4 =$ *true* and $X_2 =$ *false*, or the complements of these values. It is not hard to show that this strategy works in general.

□

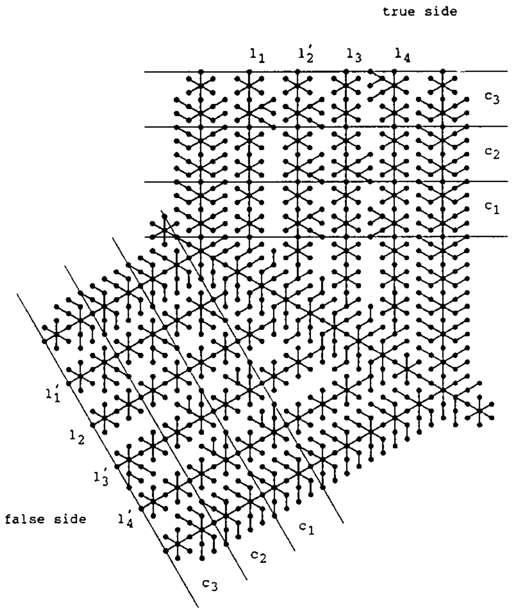


Figure 15: One realization of a transformed yes instance

Now Theorem 1 follows immediately.

In conclusion, we note that the proof of Theorem 1 constructs a tree whose only realization is on a triangular grid. Furthermore, any tree drawn on a triangular grid such that tree edges all have unit length is a minimum spanning tree. Thus our transformation can be used to show that the problem, considered in [7], of drawing a tree on a triangular grid so that every edge has unit length is NP-complete.

References

[1] P. Bose, W. Lenhart, and G. Liotta. Characterizing proximity trees. In *Graph Drawing 93*, 1993.

[2] F.J. Bradenburg. Nice drawings of graphs and trees are computationally hard. Technical Report MIP-8820, Fakultat fur Mathematik und Informatik, Univ. Passau, 1988.

[3] M. B. Dillencourt and W. D. Smith. A linear-time algorithm for testing the inscribability of trivalent polyhedra. In *Proceedings of the Eighth Annual ACM Symposium on Computational Geometry*, pages 177–185, Berlin, Germany, June 1992.

[4] M. B. Dillencourt and W. D. Smith. A simple method for resolving degeneracies in delaunay triangulations. In *Automata, Languages, and Programming: Proceedings of the 20th International Colloquium*, pages 177–188, Lund, Sweden, July 1993. Springer-Verlag Lecture Notes in Computer Science 700.

[5] P.D. Eades. Drawing free trees. *Bulletin of the Institute for Combinatorics and its Applications*, 5:10–36, 1992.

[6] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, Ca., 1979.

[7] P. J. Idicula. Drawing trees in grids. Master's thesis, Department of Computer Science, University of Auckland, 1990.

[8] E.B. Messinger, L.A. Rowe, and R.H. Henry. A divide-and-conquer algorithm for the automatic layout of large directed graphs. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-21(1):1–12, 1991.

[9] C. Monma and S. Suri. Transitions in minimum spanning trees. In *Proc. ACM Symp. on Computational Geometry*, pages 239 – 249, 1991.

[10] E. Reingold and J. Tilford. Tidier drawing of trees. *IEEE Transactions on Software Engineering*, SE-7(2):223–228, 1981.

[11] K. Sugiyama. A readability requirement on drawing digraphs: Level assignment and edge removal for reducing the total length of lines. Technical Report 45, Int. Inst. for Advanced Study of Social Information Science, March 1984.

[12] L. Valiant. Universality considerations in vlsi circuits. *IEEE Transactions on Computers*, C-30(2):135–140, 1981.

Optimal Parallel Randomized Algorithms for the Voronoi Diagram of Line Segments in the Plane and Related Problems

Sanguthevar Rajasekaran   Suneeta Ramaswami*

Department of Computer and Information Science,
University of Pennsylvania, Philadelphia, PA 19104

Abstract

In this paper, we present an optimal parallel randomized algorithm for the Voronoi diagram of a set of n non-intersecting (except possibly at endpoints) line segments in the plane. Our algorithm runs in $O(\log n)$ time with very high probability and uses $O(n)$ processors on a CRCW PRAM. This algorithm is optimal in terms of $P.T$ bounds since the sequential time bound for this problem is $\Omega(n \log n)$. Our algorithm improves by an $O(\log n)$ factor the previously best known deterministic parallel algorithm which runs in $O(\log^2 n)$ time using $O(n)$ processors [13]. We obtain this result by using random sampling at "two stages" of our algorithm and using efficient randomized search techniques. This technique gives a direct optimal algorithm for the Voronoi diagram of points as well (all other optimal parallel algorithms for this problem use reduction from the 3-d convex hull construction).

1 Introduction

Voronoi diagrams are elegant and versatile geometric structures which have applications for a wide range of problems in computational geometry and other areas. For example, computing the minimum weight spanning tree, or the all-nearest neighbor problem for a set of line segments can be solved immediately and efficiently from the Voronoi diagram of line segments. As we learnt from [17], these diagrams can also be used to compute a country's territorial waters! Certain two-dimensional motion planning problems can be solved quickly when the Voronoi diagram is available [19]. Thus, exploiting parallelism to obtain faster solutions is a desirable goal. In this paper, we are interested in developing an optimal parallel randomized algorithm on a PRAM (Parallel Random Access Machine) for the construction of the Voronoi diagram of a set of non-intersecting (except possibly at endpoints) line segments in the plane. The first sequential algorithm for this problem was given by Lee and Drysdale [17], which runs in $O(n\log^2 n)$ time. This run-time was later improved to $O(n \log n)$ in numerous papers (Kirkpatrick [15], Yap [23] and Fortune [10]), which is optimal since sorting can be reduced to this problem. The best known parallel deterministic algorithm was given by Goodrich, Ó'Dúnlaing and Yap [13] and runs in $O(\log^2 n)$ time using $O(n)$ processors. It seems unlikely that existing deterministic techniques can be used to improve this run-time.

Recent years have seen significant advances in parallel algorithm design for computational geometry problems. Some of the earliest work in this area was done by Chow [5] and Aggarwal *et. al.* [1], who gave parallel algorithms for various fundamental problems such as two-dimensional convex hulls, trapezoidal decomposition *etc.* A number of these algorithms have since been improved. Atallah, Cole and Goodrich [3] demonstrated optimal deterministic algorithms for many of these problems by applying the versatile technique of cascading divide-and-conquer and building on data structures developed in [1]. Reif and Sen [22] also obtained optimal randomized parallel algorithms for a number of these problems; these algorithms use n processors and run in $O(\log n)$ time with very high probability.

The important problems of constructing Voronoi diagrams of points in two dimensions and convex hulls of points in three dimensions, however, eluded optimal parallel solutions for a long time. Both these problems have sequential run-times of $O(n \log n)$. Aggarwal *et. al.* [1] gave $O(\log^2 n)$ and $O(\log^3 n)$ time algorithms (using n processors) for the Voronoi diagram and convex hull problems, respectively, and the technique of cascaded-merging could not be extended to these problems to improve their run-times [8]. Very recently, Goodrich [12] has given an algorithm for 3-d convex hulls that does optimal *work*, but has $O(\log^2 n)$ run-time, and Amato and Preparata [2] have described an algorithm that runs in $O(\log n)$ time but uses $n^{1+\epsilon}$ processors. Randomization, however, proves to be very useful in obtaining optimal run-time as

*This research is partially supported by ARO Grant DAAL03-89-C-0031.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

well as optimal $P.T$ bounds[1]. In [21], Reif and Sen gave an optimal randomized algorithm for the construction of the convex hull of points in three dimensions. Since the problem of finding the Voronoi diagram of points in two dimensions can be reduced to the three-dimensional convex hull problem, they also obtained an optimal parallel method for the former. Their algorithm runs in $O(\log n)$ time, using n processors, with very high probability, and there are no known deterministic algorithms that match these bounds.

2 The Use of Randomization

Recent work by Clarkson and Shor [6], Mulmuley [18], and Haussler and Welzl [14] has shown that randomization can be used to obtain better upper bounds for various geometric problems such as higher-dimensional convex hulls, halfspace range reporting, segment intersections, linear programming *etc.* Clarkson and Shor [6] used random sampling techniques to obtain tight bounds on the *expected* use of resources by algorithms for various geometric problems. The main idea behind their general technique is to use random sampling to divide the problem into smaller ones. The manner in which the random sample is used to divide the original input into subproblems depends on the particular geometric problem under consideration. They showed that for a variety of such problems, given a randomly chosen subset R of the input set S, the *expected* size of each subproblem is $O(|S|/|R|)$ and the *expected* total size is $O(|S|)$. A sample that satisfies these conditions is said to be *good*, and *bad* otherwise. They showed that any randomly chosen sample is good with constant probability, and hence bad with constant probability as well. This allows us to obtain bounds on the *expected* use of resources, but does not give high probability results (i.e. bounds that hold with probability $\geq (1 - 1/n^{\alpha})$, where n is the input size, and $\alpha > 0$). As pointed out by Reif and Sen [21], this fact proves to be an impediment in the parallel environment due to the following reason: Parallel algorithms for such problems are, typically, recursive. For sequential algorithms, since the expectation of the sum is the sum of expectations, it is enough to bound the expected run-time of each step. For recursive parallel algorithms, the run-time at any stage of the recursion will be the *maximum* of the run-times of the subproblems spawned at that stage. There is no way of determining the maximum of expected run-times without using higher moments. Moreover, even if we can bound the expected run-time at the lowest level of the recursion, this bound turns out to be too weak to bound the total run-time of the algorithm.

In [21], Reif and Sen give a novel technique to overcome this problem. A parallel recursive algorithm can be thought of as a *process tree*, where a node corresponds to a procedure at a particular stage of the recursion, and the children of that node correspond to the subproblems created at that stage. The basic idea of the technique given in [21] is to find, at every level of the process tree, a good sample with high probability. By doing this, they can show that the run-time of the processes at level i of the tree is $O(\log n/2^i)$ with very high probability, and hence the run-time of the entire algorithm is $O(\log n)$ with very high probability. By choosing a number of random samples (say $g(n)$ of them; typically $g(n) = O(\log n)$), we are guaranteed that one of them will be good with high likelihood. The procedure to determine if a sample is good or not will have to be repeated for each of the $g(n)$ samples. However, we would have to ensure that this would not cause the processor bound of $O(n)$ to be exceeded and this is done by *polling* i.e. using only a fraction of the input ($1/g(n)$, typically) to determine the "goodness" of a sample. The idea is that the assessment of a sample (good or bad) made from this smaller set is a very good reflection of the assessment that would be made from the entire set. This method finds a good sample efficiently at every level of the process tree, and is useful for converting expected value results into high probability results.

Note that the total size of the subproblems can be bound to only within a constant multiple of the original problem size. Thus in a process tree of $O(\log\log n)$ levels, this could lead to a polylogarithmic factor increase in processor bound. In [6], Clarkson and Shor get around this problem by using only a constant number of levels of recursion in their algorithm and combining this with incremental techniques (which are inherently sequential; see [6] for further details). As mentioned earlier, Reif and Sen's [21] strategy to handle this problem is to eliminate redundancy at every level of the process tree. This step is non-trivial and quite problem-specific. Our approach in this paper gets rid of this need altogether. In other words, we do not need to devise a method to control total problem size at every level of the process tree. The basic idea is to use random sampling at two stages of the algorithm. We show that the polylog factor increase in processor bound actually gives us a method to choose much larger samples, and this allows us to ignore the problem of increase in processor bound. By developing efficient search strategies to determine subproblems, we are able to obtain an optimal algorithm for the Voronoi diagram of line segments in the plane. We think that our approach is general enough to apply to other problems as well.

We would like to point out that our strategy applies to the Voronoi diagram of points in the plane as well, thus giving a direct algorithm for this problem (instead of using the reduction to 3-d convex hulls). During the course of the paper, we will point out the analogous steps for the case of points in the plane.

[1] Sorting can be reduced to these problems, and hence the best possible run-time will be $O(\log n)$ on EREW and CREW PRAMs.

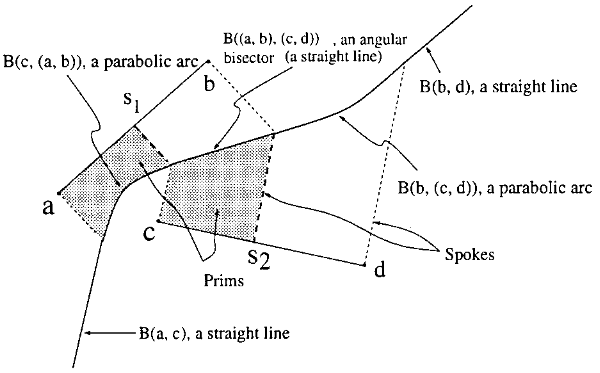

B(c, (a, b)), a parabolic arc
B((a, b), (c, d)) , an angular bisector (a straight line)
b
s1
B(b, d), a straight line
a
B(b, (c, d)), a parabolic arc
c
S2
d
Spokes
Prims
B(a, c), a straight line

Figure 1: The bisector of line segments s_1 and s_2.

3 Preliminaries and Definitions

The parallel model of computation that will be used in this paper is the *C*oncurrent *R*ead *C*oncurrent *W*rite (CRCW) PRAM. This is the sychronous shared memory model of parallel computation in which processors may read from or write into a memory cell simultaneously. Write conflicts are resolved arbitrarily; in other words, the protocol used for resolving such conflicts does not affect our algorithm. Each processor performs standard real-arithmetic operations in constant time. In addition, each processor has access to a random number generator that returns a random number of $O(\log n)$ bits in constant time. A randomized parallel algorithm $\mathcal{A}$ is said to require resource bound $f(n)$ with *very high probability* if, for any input of size n, the amount of resource used by $\mathcal{A}$ is at most $\bar{c}.\alpha.f(n)$ with probability $\geq 1 - 1/n^{\alpha}$ for positive constants $\bar{c}, \alpha$ $(\alpha > 1)$. $\tilde{O}$ is used to represent the complexity bounds of randomized algorithms i.e. $\mathcal{A}$ is said to have resource bound $\tilde{O}(f(n))$. The following is an important theorem.

Theorem 3.1 (Reif and Sen, [22]) *Given a process tree that has the property that a procedure at depth i from the root takes time T_i such that $Pr[T_i \geq k(\epsilon')^i \alpha \log n] \leq 2^{-(\epsilon')^i \alpha \log n}$, then all the leaf-level procedures are completed in $\tilde{O}(\log n)$ time, where k and α are constants greater than zero, and $0 < \epsilon' < 1$.*

Note that the number of levels in the process tree will be $O(\log \log n)$. Intuitively, the above theorem says that if the time taken at a node which is at a distance i from the root is $O((\log n)/2^i)$ with high probability, then the run-time of the entire algorithm is $\tilde{O}(\log n)$.

3.1 Voronoi Diagrams

We now give definitions and establish some notation for the geometric objects of interest in this paper i.e. Voronoi diagrams (these definitions are standard; see e.g. [13, 17]). Let S be a set of nonintersecting closed line segments in the plane. For simplicity of description, we will assume that the elements of S are in general position i.e. no three of them are cocircular. Following the convention in [17, 23], we will consider each segment $s \in S$ to be composed of three distinct objects: the two endpoints of s and the open line segment bounded by those endpoints. The Euclidean distance between two points p and q is denoted by $d(p,\ q)$. The *projection* of a point q on to a closed line segment s with endpoints a and b, denoted $proj(q,\ s)$, is defined as follows: Let p be the intersection point of the straight line containing s (call this line $\overleftrightarrow{s}$), and the line going through q that is perpendicular to $\overleftrightarrow{s}$. If p belongs to s, then $proj(q,\ s) = p$. If not, then $proj(q,\ s) = a$ if $d(q,\ a) < d(q,\ b)$ and $proj(q,\ s) = b$, otherwise. The *distance* of a point q from a closed line segment s is nothing but $d(q,\ proj(q,\ s))$. By an abuse of notation, we denote this distance as $d(q,\ s)$. Let s_1 and s_2 be two objects in S. The *bisector* of s_1 and s_2, $B(s_1,\ s_2)$, is the locus of all points q that are equidistant from s_1 and s_2 i.e. $d(q,\ s_1) = d(q,\ s_2)$. Since the objects in S are either points or open line segments, the bisectors will either be parts of lines or parabolas. The bisector of two line segments is shown in Figure 1. Note that if S is a set of points, all the bisectors are parts of straight lines. The formal definition is stated below, followed by an important theorem.

Definition 3.2 *The* Voronoi region, $V(s)$, *associated with an object s in S is the locus of all points that are closer to s than to any other object in S i.e. $V(s) = \{p \mid d(p,\ s) \leq d(p,\ s')$ for all $s' \in S\}$. The* Voronoi diagram *of S,* Vor(S), *is the union of the Voronoi regions $V(s)$, $s \in S$. The boundary edges of the Voronoi regioons are called* Voronoi edges, *and the vertices of the diagram,* Voronoi vertices.

Theorem 3.3 (Lee et al. [17]) *Given a set S of n objects in the plane (in this paper, these objects will either be nonintersecting closed line segments or points), the number of Voronoi regions, Voronoi edges, and Voronoi vertices of* Vor(S) *are all $O(n)$. To be precise, for $n \geq 3$,* Vor(S) *has at most n vertices and at most $3n - 5$ edges.*

It is convenient to divide Voronoi regions into smaller regions by augmenting the diagram in the following way (such an augmentation was originally proposed by Kirkpatrick [15]): If v is a Voronoi vertex of $V(s)$, and if $v' = proj(v,\ s)$, then the line segment obtained by joining v and v' is a *spoke* of $V(s)$. The spokes define new subregions within $V(s)$. These sub-regions bounded by two spokes, part of s and a Voronoi edge are called *primitive regions* (*prims*) [13]. Some spokes and prims are shown in Figure 1. Hereafter, all references to the Voronoi diagram will assume that it is augmented as above.

In the case of the Voronoi diagram of points, all bisectors are straight lines and hence two bisectors can intersect at most once. However for line segments, the bisectors are composed of straight line segments and parabolic arcs. Consequently, we give the following lemma, which will be of use to us later on in the paper.

Lemma 3.4 *Given line segments s, s_1 and s_2 in the plane, the two bisectors $B(s,\ s_1)$ and $B(s,\ s_2)$ can intersect at most twice.*

We give below some results that will be useful for the brute force construction of the Voronoi diagram of a sample of the appropriate size, and will be used in the course of our algorithm.

Lemma 3.5 *The Voronoi diagram of a set of n line segments in the plane can be constructed on a CREW PRAM in $O(\log n)$ time using n^3 processors.* (Proof Omited)

Lemma 3.6 *The Voronoi diagram of a set of n points in the plane can be constructed on a CREW PRAM in $O(\log n)$ using n^2 processors.*

The above lemma follows easily from the fact that intersections of n half-planes can be found in $O(\log n)$ time using n processors [1, 4]. (Amato and Preparata's [2] method, though a more complicated approach, would give us the same result.) The following are well-known non-optimal parallel algorithms that will be used when the input to a subproblem is of an appropriately small size.

Lemma 3.7 (Goodrich *et. al.*, **[13])** *The Voronoi diagram of a set of n line segments in the plane can be constructed on a CREW PRAM in $O(\log^2 n)$ time using n processors.*

Lemma 3.8 (Aggarwal *et. al.*, **[1])** *The Voronoi diagram of a set of n points in the plane can be constructed on a CREW PRAM in $O(\log^2 n)$ time using n processors.*

4 Randomized Algorithms for Voronoi Diagrams

In this section, we develop an optimal parallel randomized algorithm for the construction of the Voronoi diagram of a set of line segments in the plane. As we mentioned in Section 2, Reif and Sen [21] use the novel technique of *polling* to give an optimal randomized algorithm for the three-dimensional convex hull problem. This immediately gives an optimal randomized method for the Voronoi diagram construction of a set of points in the plane because this problem is $O(\log n)$ (parallel) time reducible to the 3-d convex hull problem. However, no analogous reduction is at hand for the case of line segments. Our optimal parallel randomized algorithm tackles the Voronoi diagram problem directly which, to our knowledge, has not been done before. The technique that we present in the forthcoming sections is general enough that it can be applied to the case of line segments as well as points in the plane. As a result, we also obtain a new and simpler randomized parallel method for the Voronoi diagram of points. Note: Even though we restrict our attention to line segments in the plane, this technique would work even when the input contains circular arcs (see [23] for relevant details).

4.1 Outline of the Method

Let $S = \{s_1, s_2, \ldots, s_n\}$ be the input set of line segments in the plane and let R be a random sample from S. Let $|R| = n^\epsilon$ for some $0 < \epsilon < 1$. The sample R will be used to divide the original input S into smaller subproblems so that each of these can be solved in parallel. The technique of using a random sample to divide the original problem into subproblems will be useful only if we can show that the subproblems are of roughly equal size ($O(n^{1-\epsilon})$) and that the total size of all the subproblems is almost equal to the size of the original problem ($O(n)$). The expected value resource bounds for computational geometry problems obtained by Clarkson and Shor [6] will be applied here. As mentioned in Section 2, due to the nature of randomized parallel algorithms it is necessary to efficiently find, at each level of the process tree, such a sample with high probability. The technique of polling developed by Reif and Sen [21] will be utilised towards this end. So a crude outline of the algorithm could be as follows.

(1). Construct the Voronoi diagram of R using the brute force technique (Lemma 3.5/Lemma 3.6). Call this diagram *Vor*(R). We use *Vor*(R) to divide the original problem into smaller problems which will be solved in parallel.

(2). Process *Vor*(R) appropriately in order to efficiently find the input set of line segments to each of these subproblems.

(3). Recursively compute (in parallel) the Voronoi diagram of each subproblem.

(4). Obtain the final Voronoi diagram from the recursively computed Voronoi diagrams.

By choosing an appropriate ϵ, we can ensure that the first step can be done in $O(\log n)$ time using n processors. In Section 4.4, we describe the use of a randomized search technique in order to efficiently find the subproblems defined by a chosen sample. We show that step 2 can be carried out in $O(\log n)$ time with high probability using n processors. In Section 4.5, we describe the merge technique to compute the Voronoi diagram from recursively computed Voronoi diagrams (this step is non-trivial) and show that the final merge step can be done in $O(\log n)$ time using n processors. Thus, the recurrence relation for the run-time is $T(n) = T(n^{1-\epsilon}) + \tilde{O}(\log n)$, which solves to $\tilde{O}(\log n)$ (this follows from Theorem 3.1). However, the description of the algorithm that we have given here is incomplete.

As mentioned earlier, there is a side-effect in such recursive algorithms that is important to consider. When we use a random sample to divide the original problem into smaller ones, we can succeed in bounding the total size of the subproblems to only within a constant multiple of n. In a recursive algorithm, this results in an increase in the total problem size as the number of recursive levels increases. For a sample size of $O(n^\epsilon)$, the depth of the process tree for a parallel randomized algorithm would be $O(\log \log n)$, and even this could result in a polylogarithmic factor increase in the total problem size. In [21], Reif and Sen get around this problem by eliminating redundancy in the input to the subproblems at every level of the recursion. In other words, since it is known that the *final* output size is $O(n)$, it is possible to eliminate those input elements from a subproblem whicch do not contribute to the final output. By doing this, they bound the total problem size at *every* level of the process tree to be within $c.n$ for some constant c. This step is non-trivial and, in general, avoiding a growth in problem size in this manner can be quite complicated. Moreover, the strategy that can be used to eliminate redundancy seems to be very problem-specific. We describe a method to use sampling at two stages of the algorithm, which will help us to overcome the problem of the increase in total input size as our algorithm proceeds down the process tree. Moreover, it appears that our strategy is general enough to be applicable to other kinds of problems as well. This technique is described in further detail in Section 4.3. It is also crucial to device efficient search strategies in order to determine *all* the subproblems that an element lies in. In our algorithm, this need is particularly important because of the fact that one of the stages of sampling entails larger sample sizes (this will become clearer in Section 4.3). We give such an efficient search strategy in Section 4.4.

4.2 Defining the Subproblems

We now give a precise description of how the Voronoi diagram of a random sample R is used to divide the original input S into smaller problems. Let $|R| = r$. Consider an edge e of *Vor*(R). e has two primitive regions (prims were defined in Section 3) on either side of it; call these $\mathcal{P}_R^1(e)$ and $\mathcal{P}_R^2(e)$. Every vertex v of *Vor*(R) is equidistant from exactly three elements of R, and v is closer to these three elements than to any other element of R. Thus every vertex defines a circle such that exactly three elements of R are incident on it and the interior of the circle is empty. Let $\mathcal{D}_R(v)$ denote the circle (and its interior) defined by Voronoi vertex v of *Vor*(R). Let v_1 and v_2 be the two vertices of e.

Consider now the region obtained by the union of $\mathcal{P}_R^1(e)$, $\mathcal{P}_R^2(e)$, and the two circles $\mathcal{D}_R(v_1)$ and $\mathcal{D}_R(v_2)$ (see Figure 2); in the case of unbounded edges, we have two prims and one circle. Call this region $\mathcal{C}_R(e)$. Observe that for the Voronoi diagram of points, it is enough to consider $\mathcal{C}_R(e)$ to be the union of just the two circles defined by the two vertices of e because the prims $\mathcal{P}_R^1(e)$ and $\mathcal{P}_R^2(e)$ always lie entirely within this union. This is not always the case for line segments, as can be seen in Figure 2.

Remark: We would like the subproblem associated with every edge of *Vor*(R) to satisfy the following condition: if the final Voronoi region $V(s_i)$ of element $s_i \in S$ intersects $\mathcal{P}_R^1(e)$ or $\mathcal{P}_R^2(e)$, then we would like at least all such s_i to be part of the input associated with the edge e.

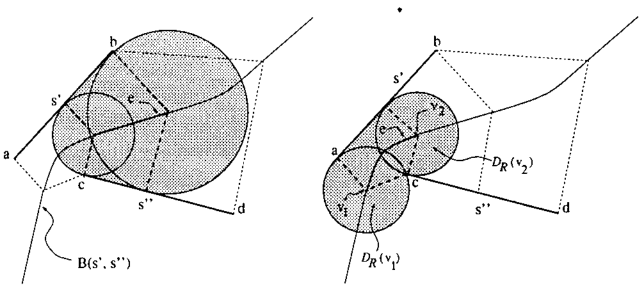


Figure 2: Region $\mathcal{C}_R(e)$ (shaded) associated with edge e.

For every element s_i of S, consider the set of points that are closer to s_i than to any other element of R. In other words, consider the Voronoi region of s_i in the Voronoi diagram of the set $R \cup \{s_i\}$. Denote this by $V^R(s_i)$. Clearly, the final Voronoi region of s_i will be a region smaller than (or equal to) $V^R(s_i)$. An element s_i will belong to the subproblem associated with edge e if and only if $V^R(s_i)$ has a non-empty intersection with $\mathcal{P}_R^1(e)$ and $\mathcal{P}_R^2(e)$. Observe that determining input for each subproblem in this manner satisfies the desired condition that we stated in the remark above. $V^R(s_i)$ intersects $\mathcal{P}_R^1(e)$ and $\mathcal{P}_R^2(e)$ if and only if *the element s_i has a non-empty intersection with $\mathcal{C}_R(e)$.*

Let $X(\mathcal{C}_R(e))$ denote the set of all elements $s_i \in S$ such that s_i has a non-empty intersection with $\mathcal{C}_R(e)$. Thus, the input set for the subproblem associated with e will be nothing but $X(\mathcal{C}_R(e))$. Each of the *Vor*($X(\mathcal{C}_R(e))$) will be computed recursively. The number of edges in *Vor*(R) will be at most $3r - 5$ (Theorem 3.3), and, hence, so will the number of regions $\mathcal{C}_R(e)$. Let $\mathcal{C}_R = \{\mathcal{C}_R(e) \mid e \text{ is a Voronoi edge from } Vor(R)\}$. We can now use Clarkson and Shor's result (Corollary 3.8, [6]) to obtain the following. Note that it takes four elements of R to define each region $\mathcal{C}_R(e)$: the two elements that e bisects, and two other elements that determine the two vertices of e (referring now to the notation used in [6], clearly $b = 4$ and the "ranges" identified by the function δ are precisely the regions $\mathcal{C}_R(e)$ defined here).

Lemma 4.1 *Given a random sample R of size r from a set of objects S of size n and using the notation established above, both of the following conditions hold with probability at least 1/2:*

$$(a) \quad \max_{\mathcal{C}_R(e) \in \mathcal{C}_R} \{ \, |X(\mathcal{C}_R(e))| \, \} \le k_{max} \, (n/r) \, \log r$$

$$(b) \quad \sum_{\mathcal{C}_R(e) \in \mathcal{C}_R} |X(\mathcal{C}_R(e))| \le k_{tot} \, (n/r) \; E(\, |\mathcal{C}_R| \,)$$

where k_{max} and k_{tot} are constants (obtained from Corollary 3.8 in [6]).

Since $E(\, |\mathcal{C}_R| \,)$ is $O(r)$, the above conditions guarantee that with probability at least 1/2, the total size of the subproblems is almost equal to the size of the original problem and the subproblems are of roughly equal size. As in [6, 21], if a sample R satisfies these conditions then it is called a *good* sample and *bad* otherwise. In order to solve the subproblems in parallel and obtain an optimal run-time, we want a good sample with *high probability.*

4.3 Two Stages of Sampling

As mentioned in Section 2, Reif and Sen [21] describe the novel technique of polling in order to obtain a good sample with high probability from samples that are only expected to be good. If $O(\log n)$ samples are chosen, instead of just one, then it follows from Lemma 4.1 that with high probability one of these samples will be good. Let R_1, R_2, $\ldots$, $R_{a\log n}$ be these samples (where the value of a is fixed according to the desired success probability of the algorithm) of size $O(n^\epsilon)$ each. Let us assume for now that we have an efficient (n processors and $O(\log n)$ time) procedure to compute the total subproblem size $\sum_{\mathcal{C}_{R_i}(e) \in \mathcal{C}_{R_i}} \mid X(\mathcal{C}_{R_i}(e)) \mid$ for each $Vor(R_i)$. However, in order to determine which of these R_i is good, this procedure will have to be repeated for each of the $O(\log n)$ samples. This will mean an increase in the processor bound of $O(n)$ which we want to avoid. Reif and Sen [21] introduce the idea of polling in order to overcome this problem. They determine the goodness of sample R_i by using only $O(n/\log^b n)$ randomly chosen elements from the input set, where $b > 2$ is some constant. The procedure to determine the goodness of the $a \log n$ samples can be run in parallel for all the R_i.

The randomized parallel algorithm is then run on the good sample found in the above manner. However the technique outlined so far still *does not* guarantee that we stay within the desired processor bound of $O(n)$. The best bound that Lemma 4.1 can give us is that the total size of the subproblems at level $(i+1)$ of the process tree is k_{tot} times the total size of the subproblems at level i. This implies that after $O(\log\log n)$ levels, the total size could increase by a polylogarithmic factor. Suppose the total size at the leaf level of the process tree is at most $n.\log^c n$, for some constant c. If the input to the divide-and-conquer algorithm were to be of size $n/\log^c n$, then the total problem size at the leaf level of the process tree would be $O(n)$. This observation (along with the results developed in the following sections) yields the following.

Theorem 4.2 *The Voronoi diagram of a set of n line segments can be constructed in $O(\log n)$ time with high probability using $n\log^c n$ processors.*

The above theorem actually enables us to choose samples of size much larger than $O(n^\epsilon)$. In particular, such a sample S' could be of size $O(n/\log^q n)$, q being a constant $> c$. If S' is a good sample, then it would again divide the original input into smaller problems of roughly equal size (i.e. $X(\mathcal{C}_{S'}(e))$ would be of size roughly $O(\log^q n)$ for all Voronoi edges e in $Vor(S')$) and $\sum |X(\mathcal{C}_{S'}(e))|$ would be $O(n)$. In order to compute the Voronoi diagram of these subproblems, we can then use any non-optimal algorithm like the one stated in Lemma 3.7 (Lemma 3.8 for points in the plane). We would, however, like to be able to find such a good sample S' with high probability. As we know from Lemma 4.1, there is a constant probability that S' is a good sample. As before, if we choose $O(\log n)$ such samples, we know that at least one of them will be good with very high probability.

Let $N = n/log^q n$. Let S_1, S_2, $\ldots$, $S_{d\log n}$ be the $O(\log n)$ samples of size N each. Since the size of S_i is large, we obviously cannot afford to construct $Vor(S_i)$ using a brute force technique (as we can do with samples of size $O(n^\epsilon)$). We will have to run the randomized parallel algorithm using each of these S_i as input. Let R_1^i, R_2^i, $\ldots$, $R_{a\log N}^i$ be the $O(\log N)$ random samples, each of size N^ϵ, chosen from S_i. Thus the skeleton of our algorithm will now be as follows. Note that the testing of the samples S_i is done with respect to a restricted input set (polling).

```
Algorithm VORONOI_DIAGRAM;
•$N := n/\log^q n$.
• Pick $d\log n$ random samples $S_1, S_2, \ldots, S_{d\log n}$ of size $N$ each.
• Let $I$ be a random subset of the input set $S$ such that $|I| = n/\log^{\bar{q}} n$, $\tilde{q}$ being a constant $<\ q$.
• $S'$ := PICK_THE_RIGHT_SAMPLE($S_1, S_2, \ldots, S_{d\log n}, I$).
• Partition the entire input $S$ according to the good sample $S'$; such a method is given in Section 4.4.
• Solve each subproblem using a non-optimal technique (Lemma 3.7 or Lemma 3.8).
• Merge the results (see Section 4.5).

Function PICK_THE_RIGHT_SAMPLE($S_1, S_2, \ldots, S_{d\log n}, I$).
• Do the following in parallel for each $S_i$ ($1 \le i \le d\log n$).
  1.  (a) Choose $a\log n$ random samples $R_1^i, R_2^i, \ldots, R_{a\log n}^i$ each of size $N^\epsilon$ from the set $S_i$.
      (b) Construct the Voronoi diagram of each $R_j^i$ ($1 \le j \le a\log N$) using the brute force technique described in Lemma 3.5 of Section 3 (Lemma 3.6 for the Voronoi diagram of points in the plane).
      (c) Determine which of these $R_j^i$ is a good sample for $S_i$.Hence the inputs to the method in Section 4.4 will be $R_j^i$ and $S_i$. Suppose $R_{j'}^i$ is one such good sample; with high probability, there will be such a $j'$.
      (d) Use $R_{j'}^i$ to divide $S_i$ into smaller subproblems.
      (e) Recursively compute (in parallel) the Voronoi diagram of each subproblem.
      (f) Obtain the final Voronoi diagram $Vor(S_i)$ from these recursively computed Voronoi diagrams.
  2. Compute the total subproblems size when restricted to $I$ (this is polling). Hence the inputs to the method in Section 4.4 will be $S_i$ and $I$.
• Return the best $S_i$; with high probability there will be such an $S_i$.
```

Observe that in the above function, it will *not* be necessary to use polling in step 1(c) because of the smaller size of the S_i. The whole point of polling is to ensure that the processor bound of $O(n)$ is not exceeded. However, we can afford to use, for all $1 \le i \le d\log n$, the whole set

S_i to determine the goodness of R^i_j ($1 \leq j \leq a\log n$) because $(d\log n).(a\log n).N$ is $o(n)$ as long as $q \geq 3$.

We know that each of the $Vor(S_i)$ can be constructed in $O(\log n)$ time with very high probability (more accurately, we will know this for sure after the next two sections). But we want to be sure that *every one of the* $Vor(S_i)$ will be constructed in $O(\log n)$ time with high probability. This follows immediately from the fact that for events A_1, A_2, ... (not necessarily disjoint), $Pr[\cup_i A_i] \leq \sum_i Pr[A_i]$ Suppose the probability that the construction of $Vor(S_i)$ takes more than β log$n$ steps is $\leq n^{-\alpha}$ for some constants α and β. Then it follows from the stated inequality that the probability that the construction of one or more of the $Vor(S_i)$ takes more than β log$n$ time is $\leq (n^{-\alpha}).(d\log n)$. Consequently, the probability that all the Voronoi diagrams $Vor(S_1)$, $Vor(S_2)$, ..., $Vor(S_{d\log n})$ are constructed in $O(\log n)$ time is $\geq (1-(n^{-\alpha}).(d\log n))$, which is very high.

It will be necessary to process the Voronoi diagram of the random sample appropriately so that we have a fast method to find the subproblems defined by it. Note that in our scheme it is imperative that we have an efficient algorithm to perform this processing. In other words, we cannot afford to have a parallel algorithm that uses a polynomial number of processors. Whereas in [21], since the sample size is always $O(n^\epsilon)$, they can choose an appropriate ϵ such that the processor bound of $O(n)$ is maintained, we do not have this flexibility. This is because of the large sample size during the first stage of sampling. In the following section, we give an efficient method that satisfies our requirement.

4.4 Finding $X(\mathcal{C}_R(e))$ for each $\mathcal{C}_R(e)$

Let R be a random sample chosen from a set $\mathcal{S}$, and let $|R| = r$. (Hence R could either be one of the S_i chosen from the original input S ($r = N$ and $\mathcal{S} = S$) or one of the R^i_j chosen from S_i ($r = N^\epsilon$ and $\mathcal{S} = S_i$).) Let us assume $Vor(R)$ is available to us; $Vor(R)$ would have been constructed either through a brute-force technique or the divide-and-conquer method outlined in the previous section. We now describe a method to process $Vor(R)$ appropriately so that we can find $X(\mathcal{C}_R(e))$ ($\subset \mathcal{S}$) for all Voronoi edges e of $Vor(R)$ in $O(\log n)$ time with high probability using $O(n)$ processors. Note that we obviously cannot sequentially determine all the subproblems that an element of $\mathcal{S}$ lies in because this could be as large as $O(r)$.

4.4.1 Processing $Vor(R)$ to Form the Search Data Structure VorDS(R)

Let U be a triangulated subdivision with u vertices. A *subdivision hierarchy* is a sequence $U_1, U_2, \ldots, U_{h(u)}$ of triangulated subdivisions satisfying the following conditions: (1) $U_1 = U$, (2) $|U_{h(u)}| = 3$, and (3) each region of U_{i+1} intersects at most τ regions of U_i for some constant τ. The idea of using fractional independent sets[2] to build a subdivision hierarchy was first proposed by Kirkpatrick [16]. Subdivision hierarchies can be used to construct efficient search data structures. Randomized parallel algorithms to solve this problem efficiently were given independently by Dadoun and Kirkpatrick [9] and by Reif and Sen [22]. We can clearly use the same idea of fractional independent sets to build a hierarchy of Voronoi diagrams that will be very useful for our purposes.

The dual $\mathcal{DV}(R)$ of the Voronoi diagram $Vor(R)$ of a set of elements R is the graph which has a node for every element in R and an edge between two nodes if their corresponding Voronoi regions share a Voronoi edge (for the case when R is a set of points, the dual is the well-known Delaunay triangulation of the set of points). $\mathcal{DV}(R)$ is planar, and can be used to build a hierarchy of Voronoi diagrams. By a slight abuse of notation, for each segment $s \in R$, we will use s to refer to its corresponding node in the graph $\mathcal{DV}(R)$ as well (hence R will refer to the set of nodes when we talk about $\mathcal{DV}(R)$). Observe that when R is a set of line segments in the plane, $\mathcal{DV}(R)$ could be a multigraph (i.e. there could be more than one edge between two nodes): this is because a bisector can be split into two or more pieces in $Vor(R)$. It can be shown that every interior face of $\mathcal{DV}(R)$ must be triangular. In a triangulated subdivision, the exterior face is assumed to be a triangle as well. If $\mathcal{DV}(R)$ does not have a triangular exterior face, we can assume that R has been augmented appropriately so that we have this property (this can be done in $O(\log r)$ time using r processors).

In the remainder of this paper, we use $V_R(s)$ to denote the Voronoi region of $s \in R$ in the Voronoi diagram $Vor(R)$. Also, as in [16], we will assume that every node of a fractional independent set is an internal node $\mathcal{DV}(R)$, and the degree of each such node is less than or equal to τ, where τ is a fixed constant (for our purposes, it suffices to assume that $\tau = 6$). Let $R_1 = R$. Consider a fractional independent set W of R_1 in the graph $\mathcal{DV}(R_1)$, and let $R_2 = R_1 - W$. Let s be a node in W. Let $\Gamma^{R_1}(s)$ represent the set of neighbors of s in $\mathcal{DV}(R_1)$ (obviously $|\Gamma^{R_1}(s)| \leq \tau$). Since W is an independent set, we know that $\Gamma^{R_1}(s) \subset R_2$. Then we have the following.

Lemma 4.3 *For an element $s' \in R_2$, $V_{R_2}(s')$ contains a part of the region $V_{R_1}(s)$ (i.e. $V_{R_2}(s') \cap V_{R_1}(s) \neq \emptyset$) if and only if $s' \in \Gamma^{R_1}(s)$.*

It is now easy to construct $Vor(R_2)$ from $Vor(R_1)$. First, we know from [9, 22] that with very high probability, we can find such an independent set W in constant time using r processors (the idea is to use *randomized*

[2] If $G = (V, E)$ is a planar graph, then a set $X \subseteq V$ is a *fractional independent set* of G if (1) the degree of each vertex $v \in X$ is less than or equal to some constant d (2) no two vertices of X are connected by an edge (i.e. the set X is an independent set) and (3) $|X| \geq c\,|V|$ for some fraction c.

symmetry breaking; see [9, 22] for details). It follows from Lemma 4.3 that $\mathcal{DV}(R_2)$ can be obtained from $\mathcal{DV}(R_1)$ in constant time using r processors: For all $s \in W$, (a) delete the node s and the edges between s and members of $\Gamma^{R_1}(s)$ and (b) add edges between pairs of elements from $\Gamma^{R_1}(s)$ if they share a Voronoi edge in $Vor(R_2)$ (there will be at most 3 such new edges for each s). Note that for each new prim of $Vor(R_2)$, we can determine in constant time the prims of $Vor(R_1)$ that it intersects.

We can repeat on R_2 the steps described above and obtain a new set R_3 and the graph $\mathcal{DV}(R_3)$, and we can continue the process until we have a set of size 3. In other words, we can build a hierarchy (analogous to the subdivision hierarchy of Kirkpatrick [16]) of Voronoi diagrams $Vor(R_1)$, $Vor(R_2)$, ..., $Vor(R_h)$ where $h = O(\log r)$. This will take $O(\log r)$ time and r processors with high probability. We build the search data structure as we construct this hierarchy of Voronoi diagrams. The data structure is a directed acyclic graph $\mathsf{VorDS}(R)$ that contains a node for each primitive region in the diagrams $Vor(R_i)$ $(1 \leq i \leq h)$. Each prim of $Vor(R_{i+1})$ is a node in the $(i+1)$-th level of $\mathsf{VorDS}(R)$ and has a directed edge to those nodes at level i with which it has a non-empty intersection. Obviously the degree of each node in the data structure is less than or equal to τ. Thus we have the following.

Lemma 4.4 *The search data structure* $\mathsf{VorDS}(R)$ *can be built in* $O(\log r)$ *time with very high probability using* r *processors, where* $r = |R|$.

4.4.2 Searching $\mathsf{VorDS}(R)$ to find $X(\mathcal{C}_R(e))$ for all $\mathcal{C}_R(e)$

Once $\mathsf{VorDS}(R)$ has been constructed, we want to search it efficiently in order to find the input set to each subproblem determined by R i.e. $X(\mathcal{C}_R(e))$ for $\mathcal{C}_R(e) \in \mathcal{C}_R$. Essentially, each element $s \in \mathcal{S}$ searches through $\mathsf{VorDS}(R)$ to determine all the subproblems that it lies in, and this is done in parallel for all such s. Note that this is also the step that allows us to determine if R is a good sample or not. (Hence in our algorithm, $\mathcal{S}$ may sometimes be a polling set that is a subset of the original input S.) Let $|\ \mathcal{S}\ | = \mathcal{N}$. We first state the basic idea behind how the search proceeds.

For each element $s \in \mathcal{S}$, we start off the search by determining the prims at level h that $V^{R_h}(s)$ intersects[3]. Obviously $V^{R_h}(s)$ is of constant size and hence this step can be carried out in constant time using $O(\mathcal{N})$ processors. Our algorithm works in phases. We describe the steps during one phase: Suppose that for $s \in \mathcal{S}$, a processor Π has reached the node corresponding to prim $\mathcal{P}$ at level $(i+1)$ of $\mathsf{VorDS}(R)$. As we know from the previous section, $\mathcal{P}$ intersects a constant number of prims at level i. Let these be called $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_\rho$ (note that ρ will be less than or equal to τ). Without loss of generality, let s_1, s_2, ..., s_ρ be the elements of R_i such that for $1 \leq j \leq \rho$, $\mathcal{P}_j$ belongs to $V_{R_i}(s_j)$, respectively. If $B(s,\ s_j)$ intersects prim $\mathcal{P}_j$, then Π generates a request for a new processor in the following manner: it randomly generates a processor label and writes into that processor to indicate that it is being requested. There could be write conflicts during this step and an arbitrary processor will succeed (in the case of a success, the new processor will need to know $\mathcal{P}_j$ and s so as to continue the search from the node corresponding to $\mathcal{P}_j$). Obviously Π can carry out this step in constant time (ρ steps). In order to be sure that this scheme works, we need to know that for each $s \in \mathcal{S}$, every prim that should be reached at level i is reachable from level $(i+1)$ in the above described manner. This follows from the lemma given below (we omit the proof, which is quite straightforward).

Lemma 4.5 *If* $V^{R_i}(s)$ *intersects a prim* $\mathcal{P}$ *at level* i, *then* $V^{R_{i+1}}(s)$ *must intersect at least one of the prims that* $\mathcal{P}$ *intersects at level* $(i+1)$ *of* $\mathsf{VorDS}(R)$.

We described one phase of the algorithm above. During the next phase, another set of requests are generated in a similar manner (including the ones that weren't satisfied in the previous phase), and so on. It follows from the above lemma that by searching through $\mathsf{VorDS}(R)$ in the described manner, we can find $V^R(s)$ and hence all the subproblems that s lies in.

There is an important fact that we have ignored so far. In order to carry out this search step efficiently, we have to be sure that the total number of requests generated is $O(\mathcal{N}\log\mathcal{N})$ over all levels of $\mathsf{VorDS}(R)$. Observe that it is possible for the total problem size to increase from level $i = 1$ to level $i = h$ of $\mathsf{VorDS}(R)$ because an element $s \in \mathcal{S}$ might lie in a greater number of subproblems at level $i+1$ than at level i. However, we can prove that this increase is such that the total subproblems size is still $O(\mathcal{N})$ at every level of $\mathsf{VorDS}(R)$. We will not go into the proof, for lack of space and appeal to intuition instead. Consider an element $s' \in R_1$ such that s' belongs to the independent set W of $\mathcal{DV}(R_1)$. For every element that belongs to $\Gamma^{R_1}(s')$ (the neighbors of s' in $\mathcal{DV}(R_1)$), the total problem size associated with that element at level 2 of $\mathsf{VorDS}(R)$ could increase. This is because elements that belong to the subproblems associated with s' (there can be at most a constant number of such subproblems) could, upon the removal of s', now belong to the neighbors as well; by an abuse of notation, call this set of elements $X(\mathcal{C}_{R_1}(s'))$. Hence the maximum possible increase in size at level 2 of $\mathsf{VorDS}(R)$ will be given by $\sum_{s' \in W} |\Gamma^{R_1}(s')|.|X(\mathcal{C}_{R_1}(s'))|$. In a similar manner, we can compute the total increase at level 3 of the data structure, while taking into account the possible increase in the problem size associated with an independent set element at level 2. We are able to prove that this increase at each level of $\mathsf{VorDS}(R)$ is given by a

[3] Recall that we use $V^R(s)$ to denote the region that is closer to s than to any other element in R i.e. the Voronoi region of s in the Voronoi diagram of $R \cup \{s\}$.

geometric series of ratio less than 1, as long as the fraction of independent set elements at each level of the data structure is less than μ, where $\mu < 1$ is fixed. We can show that this must be the case with very high probability. The intuition behind this result is as follows: Those independent set elements that cause a large increase in problem size must have high degree in $\mathcal{DV}(R)$, and those that cause a smaller increase will have a smaller degree. Hence there will be fewer elements that cause a large increase than those that cause a smaller increase (since in a planar graph there will be fewer vertices of large degree than vertices of smaller degree).

The search through this data structure is carried out by the dynamic allocation of processors to each segment $s \in \mathcal{S}$. The idea is to use a number of processors proportional to $|Y^{R_1}(s)|$ for s. We can use randomized techniques for dynamic processor allocation (such methods are standard, as given, for example, in [20]). For good samples, all the requests will be satisfied in $O(\log \mathcal{N})$ phases with very high likelihood using $O(\mathcal{N})$ processors. If the search through $\mathsf{VorDS}(R)$ is incomplete after these $O(\log \mathcal{N})$ phases (i.e. there are requests that haven't been satisified), then we can reject R with high confidence.

Note that since the data structure that we are searching through is a directed acyclic graph, it is possible that an $s \in \mathcal{S}$ may arrive at the same node through two different paths. In such instances, we want to avoid repeating searches that have already commenced. We would like to use $O(\mathcal{N} \log \mathcal{N})$ space in order to solve this problem; we can use randomized hashing techniques to achieve this goal. This will use $O(\mathcal{N})$ space for each level of $\mathsf{VorDS}(R)$. From Gil, Matias and Vishkin [11], we know that we can certainly do this in $O(\log^* \mathcal{N})$ time (nearly constant) with high probability by doing an optimal amount of work at each phase. But we show that over all the levels of the data structure, the total processing time is $O(\log \mathcal{N})$ with high probability (omitted for lack of space). Thus we have the following.

Lemma 4.6 *We can find $X(\mathcal{C}_R(e))$ for all $\mathcal{C}_R(e) \in \mathcal{C}_R$ in $\tilde{O}(\log \mathcal{N})$ time using $O(\mathcal{N})$ processors and $\tilde{O}(\mathcal{N} \log \mathcal{N})$ space, where R is a random sample of S and $|S| = \mathcal{N}$.*

4.5 Merging Recursively Computed Voronoi Diagrams

In the previous section, we described the method to find the subproblems determined by a random sample from the input set $\mathcal{S}$. So assume now that for a *good* sample R, $X(\mathcal{C}_R(e))$ has been found for all $\mathcal{C}_R(e) \in \mathcal{C}_R$. The diagrams $Vor(X(\mathcal{C}_R(e)))$ will be computed recursively, in parallel for each $\mathcal{C}_R(e)$. We want to merge these Voronoi diagrams to form $Vor(\mathcal{S})$. In the remainder of the section, we describe the method to perform this merge step efficiently.

The final Voronoi diagram of $\mathcal{S}$ can be constructed by computing the final Voronoi region $V(s)$ for each element $s \in \mathcal{S}$. $V(s)$ can be determined by finding the *intersection* of all the Voronoi regions of s that have been computed recursively in the subproblems. As before, let $|R| = r$ and let $|\mathcal{S}| = \mathcal{N}$. Since R is a good sample of $\mathcal{S}$, we know that the total size of the subproblems is less than or equal to $k_{tot}\mathcal{N}$. It follows therefore that the *total* number of Voronoi edges in *all* the recursively computed Voronoi diagrams must be less than $3.k_{tot}\mathcal{N}$ (since the number of Voronoi edges in $Vor(X(\mathcal{C}_R(e)))$ is less than $3.|X(\mathcal{C}_R(e))|$).

Consider the set of subproblems in which $s \in \mathcal{S}$ lies. Let $\Upsilon_R(s)$ be the set of recursively computed Voronoi regions of s in these subprobles. That is[4], $\Upsilon_R(s) = \{V_{X(\mathcal{C}_R(e))}(s) \mid X(\mathcal{C}_R(e) \in \mathcal{C}_R\}$. The Voronoi regions will be represented as the collection of their Voronoi edges. Note that the total size of all the $\Upsilon_R(s)$ for all $s \in \mathcal{S}$ will also be $O(\mathcal{N})$ since every Voronoi edge is counted exactly two times over the regions $\Upsilon_R(s)$ for all $s \in \mathcal{S}$ (once for each of the two Voronoi regions that it borders). Every Voronoi edge in $\Upsilon_R(s)$ is part of some bisector $B(s, s')$ $(s' \in \mathcal{S})$. In the case of line segments, computing the intersection of these bisectors to find $V(s)$ is not an efficient strategy because of the fact that two such bisectors can intersect twice (see Lemma 3.4). The following lemma provides us with a solution. (We omit the proof for lack of space.)

Lemma 4.7 *Every Voronoi vertex of the final Voronoi region $V(s)$ appears as a vertex in at least one of the Voronoi regions in the set $\Upsilon_R(s)$.*

Thus we just have to search the set of vertices that appear in the Voronoi regions in $\Upsilon_R(s)$ and eliminate those vertices that are not part of $V(s)$. We do this as follows. The vertices of any Voronoi region of an element s have well-defined sorted order defined by the projection of the vertices on s (this is given as the function *proj* in Section 3). Assume that the vertices appear in clockwise order (with respect to some axis that is fixed on s). We merge in parallel the Voronoi vertices of the regions in $\Upsilon_R(s)$ according to this ordering by using the cascading divide-and-conquer technique that sorts optimally [Cole [7],Atallah, Cole and Goodrich [3]], until finally we are left with the vertices of $V(s)$. Each leaf of the merge tree will contain a Voronoi vertex from the regions in $\Upsilon_R(s)$ along with the Voronoi edge to its right. Let y_s be the total number of Voronoi *edges* in $\Upsilon_R(s)$.

We say that a Voronoi vertex v is *hidden* by a Voronoi edge e if the segment from v to $proj(v, s)$ intersects e and v does not lie on e. The idea is to maintain, at each node κ of the merge tree and each phase i of the cascade, a list that consists of vertices that are not hidden by any of the Voronoi edges encountered at κ until phase i. After the final phase, the root of the merge tree will have the list

[4] Recall that we use $V_A(a)$ to denote the Voronoi region of $a \in A$ in the diagram $Vor(A)$.

of Voronoi vertices that belong to $V(s)$. This step runs in $O(\log\ y_s)$ time with $O(y_s)$ processors. We execute this step in parallel for all $s \in \mathcal{S}$ and since $\sum_{s \in \mathcal{S}} y_s$ is $O(\mathcal{N})$, we have the following.

Lemma 4.8 *The Voronoi diagram* $\mathrm{Vor}(\mathcal{S})$ *can be constructed from the recursively computed Voronoi diagrams* $\mathrm{Vor}(X(\mathcal{C}_R(e)))$ *(*$\mathcal{C}_R(e) \in \mathcal{C}_R$*) in* $O(\log \mathcal{N})$ *time using* $O(\mathcal{N})$ *processors.*

5 Conclusions and Applications

It follows from the previous section that we have an optimal algorithm for the Voronoi diagram of a set of line segments in the plane. We state this below.

Theorem 5.1 *The Voronoi diagram of a set of* n *non-intersecting line segments in the plane can be computed in* $\tilde{O}(\log n)$ *time using* $O(n)$ *processors and* $\tilde{O}(n \log n)$ *space.*

The above theorem immediately gives us optimal parallel randomized algorithms for computing the minimum weight spanning tree, the all-nearest neighbor for the set of segments and also an optimal parallel algorithm to plan the motion of an object from one point to another while avoiding polygonal obstacles (see [19] for details).

Clarkson and Shor's [6] results established the effectiveness of random sampling in deriving better expected bounds for computational geometry problems and Reif and Sen [21] demonstrate how to obtain high probability results from expected bounds, which is crucial for parallel algorithms. Our result offers more evidence of the usefulness of randomization in obtaining more efficient algorithms. It will be interesting to see if our technique can be applied to other problems to obtain more efficient results and perhaps less complicated algorithms since we believe that our approach is simpler and more general. For instance, we think that the 3-d convex hull algorithm given by Reif and Sen [21] could be simplified, especially since the surface of a convex polyhedron is topologically equivalent to a bounded planar subdivision.

References

[1] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, and C. K. Yap. Parallel Computational Geometry. *Algorithmica*, 3:293–327, 1988.

[2] N. M. Amato and F. P. Preparata. An NC^1 Parallel 3D Convex Hull Algorithm. In *Proc. 9th ACM Symp. on Computational Geometry*, 1993.

[3] M. J. Atallah, R. Cole, and M. T. Goodrich. Cascading Divide-and-Conquer: A Technique for Designing Parallel Algorithms. *SIAM J. Comput.*, 18(3):499–532, 1989.

[4] M. J. Atallah and M. T. Goodrich. Efficient Parallel Solutions to Geometric Problems. In *Proc. 1985 IEEE Conf. on Parallel Processing*, pages 411–417, 1985.

[5] A. Chow. *Parallel Algorithms for Geometric Problems.* PhD thesis, Univ. of Illinois at Urbana-Champaign, 1980.

[6] K. L. Clarkson and P. W. Shor. Applications of Random Sampling in Computational Geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[7] R. Cole. Parallel Merge Sort. *SIAM J. Comput.*, 17(4):770–785, 1988.

[8] R. Cole and M. T. Goodrich. Optimal Parallel Algorithms for Polygon and Point-set Problems. *Algorithmica*, 7:3–23, 1992.

[9] N. Dadoun and D. Kirkpatrick. Parallel Processing for Efficient Subdivision Search. In *Proc. Third ACM Symp. on Computational Geometry*, pages 205–214, 1987.

[10] S. Fortune. A Sweepline Algorithm for Voronoi Diagrams. In *Proc. 2nd ACM Symp. on Computational Geometry*, pages 313–322, 1986.

[11] J. Gil, Y. Matias, and U. Vishkin. Towards a Theory of Nearly Constant Time Parallel Algorithms. In *Proc. of Symp. on FOCS*, 1992.

[12] M. T. Goodrich. Geometric Partitioning Made Easier, Even in Parallel. In *Proc. 9th ACM Symp. on Computational Geometry*, 1993.

[13] M. T. Goodrich, C. Ó'Dúnlaing, and C. K. Yap. Constructing the Voronoi Diagram of a Set of Line Segments in Parallel. In *Lecture Notes in Computer Science: 382, Algorithms and Data Structures, WADS*, pages 12–23. Springer-Verlag, 1989.

[14] D. Haussler and E. Welzl. ϵ-nets and Simplex Range Queries. *Discrete and Computational Geometry*, 2:127–152, 1987.

[15] D. G. Kirkpatrick. Efficient Computation of Continuous Skeletons. In *Proc. 20th IEEE Symp. on Foundations of Computer Science*, pages 18–27, 1979.

[16] D. G. Kirkpatrick. Optimal Search in Planar Subdivisions. *SIAM J. Comput.*, 12(1):28–35, 1983.

[17] D. T. Lee and R. L. Drysdale. Generalization of Voronoi Diagrams in the Plane. *SIAM J. Comput.*, 10(1):73–87, February 1981.

[18] K. Mulmuley. A Fast Planar Partition Algorithm. In *Proc. 20th IEEE Symp. on the Foundations of Computer Science*, pages 580–589, 1988.

[19] C. Ó'Dúnlaing and C. K. Yap. A 'Retraction' Method for Planning the Motion of a Disc. *J. Algorithms*, 6:104–111, 1985.

[20] J. H. Reif and S. Sen. Polling: A New Randomized Sampling Technique for Computational Geometry. Manuscript.

[21] J. H. Reif and S. Sen. Optimal Parallel Randomized Algorithms for Three Dimensional Convex Hulls and Related Problems. *SIAM J. Comput.*, 21(3):466–485, 1992.

[22] J. H. Reif and S. Sen. Optimal Randomized Parallel Algorithms for Computational Geometry. *Algorithmica*, 7:91–117, 1992.

[23] C. K. Yap. An $O(n \log n)$ Algorithm for the Voronoi Diagram of a Set of Simple Curve Segments. *Discrete and Computational Geometry*, 2:365–393, 1987.

Constructing Levels in Arrangements and Higher Order Voronoi Diagrams*

Pankaj K. Agarwal[1] Mark de Berg[2] Jiří Matoušek[3]
Otfried Schwarzkopf[2]

Abstract

We give a simple lazy randomized incremental algorithm to compute $\leq k$-levels in arrangements of x-monotone Jordan curves in the plane, and in arrangements of planes in three-dimensional space. If each pair of curves intersects in at most s points, the expected running time of the algorithm is $O(k^2\lambda_s(n/k) + \min(\lambda_s(n)\log^2 n, k^2\lambda_s(n/k)\log n))$. For the three-dimensional case the expected running time is $O(nk^2 + \min(n\log^3 n, nk^2 \log n))$. The algorithm also works for computing the $\leq k$-level in a set of discs, with an expected running time of $O(nk + \min(n\log^2 n, nk\log n))$. Furthermore, we give a simple algorithm for computing the order-k Voronoi diagram of a set of n points in the plane that runs in expected time $O(k(n-k)\log n + n\log^3 n)$.

1 Introduction

Arrangements of hyperplanes have been studied for a long time in combinatorial and computational geometry and yet they have kept some of their secrets. Some of the intriguing open questions are related to the concept of *levels*. We say that a point p is at level k with respect to a set H of non-vertical hyperplanes in $\mathbb{R}^d$ if there are exactly k hyperplanes in H that lie strictly above p. The k-level of an arrangement $\mathcal{A}(H)$ of hyperplanes is the closure of all facets of $\mathcal{A}(H)$ whose interior points have level k with respect to H—see Figure 1. The $\leq k$-level of $\mathcal{A}(H)$ is the union of all i-levels of $\mathcal{A}(H)$, for $0 \leq i \leq k$. The k-level and the $\leq k$-level of arrangements of monotone surfaces are defined analogously. In fact, one can give a more general definition of levels. Given a family Γ of subsets, also called *ranges*, of $\mathbb{R}^d$, define the level of a point p with respect to Γ to be the number of ranges that contain p in their interior; the k-level and $\leq k$-level in the arrangement of Γ are defined as before. For a set H of hyperplanes, if we choose ranges to be the half-spaces lying above the hyperplanes of H, then the level of p is same under the two definitions.

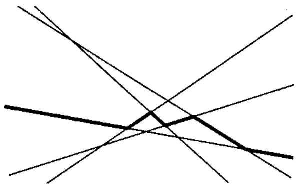


Figure 1: The 2-level in an arrangement of lines.

Although many researchers have studied combinatorial aspects of levels in arrangements, the maximum complexity of the k-level is still unknown. Even in the plane there is a large gap between the known upper and lower bound on the maximum complexity of the k-level: the best known lower bound is $\Omega(n\log(k+1))$ [Ede87], whereas the best known upper bound is $O(n\sqrt{k}/\log^*(k+1))$ [PSS92]. However, exact bounds are known for the maximum complexity of the $\leq k$-level in an arrangement of hyperplanes

*Work on this paper by P.A. has been supported by National Science Foundation Grant CCR–93–01259 and an NYI award. M.d.B. and O.S. acknowledge support by the Netherlands' Organization for Scientific Research (NWO) and partial support by ESPRIT Basic Research Action No. 7141 (project ALCOM II: *Algorithms and Complexity*). Work by J.M. was supported by Charles University Grant No. 351 and by Czech Republic Grant GAČR 201/93/2167. Part of this research was done when P.A. and O.S. visited Charles University, and when P.A. visited Utrecht University. These visits were supported by Charles University and NWO.

[1] Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA.

[2] Vakgroep Informatica, Universiteit Utrecht, Postbus 80.089, 3508 TB Utrecht, the Netherlands.

[3] Katedra aplikované matematiky, Universita Karlova, Malostranské nám. 25, 118 00 Praha 1, Czech Republic.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

in $\mathbb{R}^d$. Using a probabilistic counting method, Clarkson and Shor [CS89] proved that the maximum complexity of the $\leq k$-level is $\Theta(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$. Following a similar technique, Sharir [Sha91] proved that the maximum complexity of the $\leq k$-level in a set of x-monotone Jordan curves, with at most s pairwise intersections, is $\Theta(k^2 \lambda_s(n/k))$. Here $\lambda_s(m)$ is the maximum length of an (m, s)-Davenport-Schinzel sequence; for any constant s, $\lambda_s(m)$ is roughly linear in m [ASS89]. For discs he proved a bound of $\Theta(nk)$ on the complexity of the $\leq k$-level.

On the algorithmic side, Mulmuley [Mul91b] gave a randomized incremental algorithm for computing the $\leq k$-level in hyperplane arrangements in any dimension. For $d \geq 4$, the expected running time of his algorithm is $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$, which is optimal. For $d = 2, 3$ the expected running time of his algorithm is $O(nk^{\lceil d/2 \rceil} \log(n/k))$, which is suboptimal by a logarithmic factor. Recently, Everett et al. [ERvK93] gave an optimal $O(n \log n + nk)$ expected time randomized algorithm for computing the $\leq k$-level in an arrangement of lines in the plane. Mulmuley's algorithm can be applied to compute the $\leq k$-level of arrangements of x-monotone Jordan curves in the plane, but there does not seem to be an easy way to generalize it for computing the $\leq k$-level in more general ranges like discs. Sharir [Sha91] presented a divide-and-conquer algorithm for computing the $\leq k$-level of rather general ranges in the plane; its worst-case running time is roughly $\log^2 n$ times the maximum size of the $\leq k$-level.

In this paper we show that by modifying Mulmuley's algorithm slightly, we can obtain a simpler and faster randomized algorithm for computing the $\leq k$-level in a hyperplane arrangement whose expected running time is $O(nk^{\lceil d/2 \rceil} + n \log n \min(k^{\lceil d/2 \rceil}, \log^{\lceil d/2 \rceil} n))$, which is optimal for $k \geq \log^2 n$ if $d = 2$, for $k \geq \log^{3/2} n$ if $d = 3$, and for all values of k if $d \geq 4$. Moreover, the modified algorithm works for more general ranges like discs or pseudo-discs as well.

Let us have a brief look at Mulmuley's algorithm [Mul91b] (see also his book [Mul93]). The n hyperplanes are added one by one in random order and during this process the $\leq k$-level of the current arrangement is maintained. For all faces of the current structure a conflict list (containing the not yet inserted hyperplanes that intersect the face) is maintained. When adding a new hyperplane h, the algorithm first updates the arrangement locally at faces that are intersected by h. Then it removes cells that have "fallen off" because they are now on level $k + 1$; Mulmuley calls this the peeling step.

We observe that this algorithm maintains too much information. After adding the first $n/2$ hyperplanes we expect the k-level of $\mathcal{A}(H)$ to be located near the $k/2$-level of $\mathcal{A}(R)$, where R is the set of the first $n/2$ hyperplanes; most of the $\leq k$-level of $\mathcal{A}(R)$ lying below this $k/2$-level will be peeled off in later stages. Hence, we only maintain the $\leq \kappa(r)$-level of the first r hyperplanes, where $\kappa(r)$ is approximately $k(r/n)$. However, now we have to be a bit more careful about discarding layers of cells. It turns out that the easiest way to do this is to remove the peeling step altogether, and to replace it by regular clean-up phases. This idea was inspired by the lazy randomized incremental algorithms of de Berg et al. [dBDS94]. At the same time, this makes it possible to use the algorithm in situations where it is not obvious how to access the parts of the current cell complex that must be peeled off. One such situation is for the $\leq k$-level for discs.

As stated earlier, not much is known about the complexity of the exact k-level. There is one special case where the maximum complexity of the k-level of a three-dimensional arrangement is known to be $\Theta(k(n - k))$, namely when all the hyperplanes are tangent to the unit paraboloid. This situation arises when the planes are the images of a set of points in the plane under the transformation that maps the order-k Voronoi diagram of these points to the k-level of the planes. Most known algorithms for computing the k-levels in three-dimensional space actually compute the $\leq k$-level [Mul91b, CE87, BDT93]. Since the complexity of the $\leq k$-level is $\Theta(nk^2)$ in the situation sketched above, the running time of these algorithms is $\Omega(nk^2 + n \log n)$. The randomized incremental algorithm by Aurenhammer and Schwarzkopf [AS92] maintains only the k-level, but it can be shown that any randomized incremental algorithm that maintains the k-level of the intermediate arrangements must take time $\Omega(nk^2)$ as well, since the expected number of structural changes in the k-level is $\Omega(nk^2)$ [AS92]. The only algorithm that approaches the desired $O(k(n - k))$ time bound was presented by Clarkson [Cla87]. His algorithm runs in time $O(n^{1+\varepsilon} k)$, where $\varepsilon > 0$ is an arbitrarily small constant. (The algorithm can probably be improved somewhat by using more recent results on geometric cuttings).

We present a simple randomized incremental algorithm that runs in $O(k(n - k) \log n + n \log^3 n)$ time. We obtain this result by maintaining neither the entire $\leq k$-level nor the exact k-level of the current arrangement, but some suitably chosen region that is guaranteed to contain the k-level of the full set.

2 Computing the $\leq k$-level

Let H be a set of n hyperplanes in d-dimensional space. We denote the arrangement formed by H as $\mathcal{A}(H)$ and the level of a point p with respect to H as $\ell_H(p)$. We define the level of a face of $\mathcal{A}(H)$ as the level of any of its interior points. For a subset $R \subset H$, we denote the *bottom-vertex triangulation* of the $\leq k$-level of $\mathcal{A}(R)$ by $\mathcal{L}_k(R)$. This triangulation is defined as follows. Let $\mathcal{C}$ be a cell of the $\leq k$-level and let v be the bottom vertex of $\mathcal{C}$ (that is, the lexicographically smallest vertex of the cell). If $d = 1$ then $\mathcal{C}$ is a segment and it is already a (1-dimensional) simplex. If $d > 1$, then we recursively triangulate the $(d-1)$-dimensional facets of $\mathcal{C}$ and extend each $(d-1)$-simplex into a d-simplex using vertex v. The bottom-vertex triangulation $\mathcal{L}_k(R)$ of the $\leq k$-level of $\mathcal{A}(R)$ is obtained by triangulating every cell of the $\leq k$-level in this manner. (Unbounded cells of $\mathcal{A}(R)$ require some care in this definition [Cla88].) Observe that the bottom-vertex triangulation is a cell-complex: any facet of a simplex is shared with exactly one other simplex. In this section we give an algorithm to compute $\mathcal{L}_k(H)$ for a given parameter $k < n$.

The algorithm. To compute $\mathcal{L}_k(H)$ we first generate a random permutation $h_1, h_2, \ldots, h_n$ of H. Let $H_r := \{h_1, \ldots, h_r\}$, and let $\kappa(r)$ be an appropriate non-decreasing function with $\kappa(n) \geq k$ to be defined below. The idea of the algorithm is to maintain $\mathcal{L}_{\kappa(r)}(H_r)$ while adding the hyperplanes one by one. However, it turns out to be easier not to discard cells whose level (with respect to H_r) has grown too large after every step. Instead we get rid of these cells at regular *clean-up*-phases, namely after inserting the 2^i-th hyperplane, for $1 \leq i \leq \lfloor \log n \rfloor$, and after inserting the last hyperplane. This type of *lazy randomized incremental algorithm* was introduced recently by de Berg et al. [dBDS94]. Below we give a detailed description of the algorithm.

We maintain a collection of d-simplices that forms a d-dimensional cell complex, defined as follows. Suppose we have added the first r hyperplanes. Then the collection $\mathcal{S}_r$ forms the bottom-vertex triangulation of the cells of $\mathcal{A}(H_r)$ whose level is at most $\kappa(p)$ with respect to H_p, where p is the time when the latest clean-up was performed. Our schedule for performing clean-ups implies that p is the largest power of two that is less than r. With every simplex $\Delta \in \mathcal{S}_r$ we keep a *conflict list* $H(\Delta)$ that contains the hyperplanes in H intersecting the interior of Δ, and with each hyperplane $h \in H \setminus H_r$ we keep a list of simplices of $\mathcal{S}_r$ that h intersects. We also keep an adjacency graph $\mathcal{G}$ on the simplices. Two simplices Δ_1 and Δ_2 are connected in this graph if they share a facet. We also mark an arc of this graph if the shared facet of the two simplices connected by this arc is contained in a hyperplane. The level of points changes if and only if we pass from one simplex to an adjacent one connected by a marked arc.

Now consider the addition of hyperplane h_{r+1}. In a generic step of the algorithm (that is, when $r+1$ is not a power of 2) we must perform two tasks: update our collection of simplices (that means, re-triangulate the cells that are split by h_r) and their adjacency relations, and compute the conflict lists for the new simplices.

Let us start with the first task. Using the conflict lists we identify all the simplices in $\mathcal{S}_r$ that are intersected by h_{r+1}; those are the only ones influenced by the insertion of h_{r+1}. Because we know the adjacency relations among the simplices, we can get the intersected simplices in a number of groups, one for each cell that is intersected. Consider such a group, and let $\mathcal{C}$ be the corresponding cell. All the simplices in the group have to be deleted and replaced by a number of new simplices. Let us first take a look at the situation when $d = 2$—see Figure 2. To deal with the part of $\mathcal{C}$ that lies on the same side of h_{r+1} as the bottom vertex v of $\mathcal{C}$, we draw new diagonals from v to the two intersection points of h_{r+1} with the boundary of $\mathcal{C}$; this creates three new simplices. The part of $\mathcal{C}$ lying on the opposite side of h_{r+1} is simply triangulated from scratch. This way the bottom-vertex triangulations of the two cells that result from the splitting of $\mathcal{C}$ are constructed in time that is linear in the number of new simplices that are created. The adjacency relations among the simplices can easily be determined during the process. In dimensions greater than two things are not very different. Consider a 3-dimensional cell $\mathcal{C}$. We first re-triangulate the 2-dimensional facets that are intersected, in the way we just described. We also triangulate $\mathcal{C} \cap h_{r+1}$. Next we construct the 3-dimensional simplices. Given the triangulation of the facets of the cell, the procedure for this is analogous to the planar procedure. In general, in dimension d we first treat the parts of the $(d-1)$-facets incident to intersected simplices recursively, and then we extend the $(d-1)$-simplices that we get to d-simplices by connecting them to the correct bottom vertex.

Our second task is to compute the conflict lists of the new simplices. Let $\mathcal{C}$ be a cell that has been split by h_{r+1}, and let $\mathcal{C}'$ be one of the two new cells contained in $\mathcal{C}$. Some of the simplices of $\mathcal{C}'$ may already have existed in $\mathcal{C}$; these simplices already have the correct conflict lists. To find the conflict lists for the new simplices we collect all the hyperplanes from old

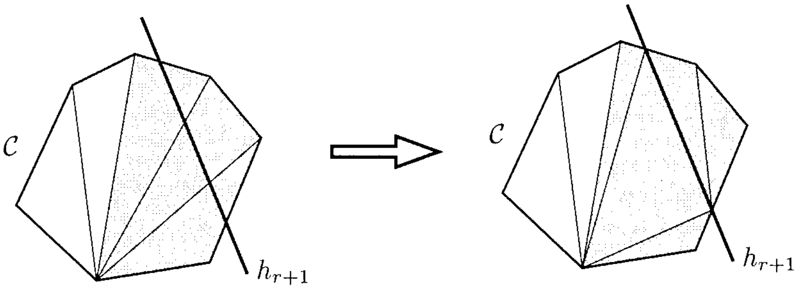


Figure 2: Re-triangulation of a cell that is split.

simplices that intersect $\mathcal{C}'$. We remove the duplicates from our collection, and for each hyperplane we determine one new simplex that it intersects (its *initial simplex*). This can be done in time proportional to the total size of the conflict lists involved. Next, we perform for each hyperplane a traversal of the adjacency graph on the new simplices, starting from its initial simplex. This way we determine all the new simplices intersected by the hyperplane. After having done this for each hyperplane, we have constructed all the new conflict lists. The time we have spent is linear in the size of these lists plus the size of the old lists that disappear.

In a generic step of the algorithm this is all we do. (It would be difficult to discard simplices at every step. The reason for this is the following. Since our "threshold" $\kappa(r)$ depends on r, it is not necessarily the case that all simplices lying below the new hyperplane h_{r+1} can be removed. Therefore we would have to maintain the levels of the simplices, so that we can decide in later stages whether or not to discard them. But maintaining the level of the simplices explicitly is too costly, because it involves simplices not intersected by h_{r+1}.) For $r = 2, 4, \ldots, 2^{\lfloor \log n \rfloor}, n$, however, we perform a clean-up step. We traverse the entire current cell complex using the adjacency graph $\mathcal{G}$. While visiting each simplex we compute its level and discard it if its level is larger than $\kappa(r)$, where r is the current step. There will be a slight chance that by doing so we discard a cell that later turns out to contain a part of the $\leq k$-level of $\mathcal{A}(H)$, but we will argue that the probability that this happens is small. If it happens, we will notice so in the final clean-up phase and we will restart the algorithm with a new random permutation. Since the probability of picking such an unfortunate permutation is small, this procedure changes the expected running time of the algorithm by a constant factor only.

The analysis. Consider the situation after step r of the algorithm. Let p be the largest power of two that is less than r, and define $k^* := \kappa(p)$. From now on we assume that r and, hence, p and k^* are fixed.

We use the abstract framework of de Berg et al. [dBDS94] to analyze our algorithm. Let $\mathcal{F}$ be the set of all possible simplices Δ, that is, all Δ that arise in the bottom-vertex triangulation of $\mathcal{A}(R)$ for some subset $R \subseteq H$. We denote the defining set of a simplex $\Delta \in \mathcal{F}$ by $\mathcal{D}(\Delta)$; this is the inclusion-minimal subset R for which Δ appears in the bottom-vertex triangulation of $\mathcal{A}(R)$. (In degenerate situations, different defining sets can correspond to simplices with the same set of vertices; these simplices are considered to be distinct.) We define the killing set of Δ, denoted $\mathcal{K}(\Delta)$, as the set of hyperplanes in H that intersect the interior of Δ. Furthermore, for a subset $R \subseteq H$ we define $\mathcal{T}(R)$ as the set of simplices in the bottom-vertex triangulation of the full arrangement $\mathcal{A}(R)$, and we define $\mathcal{M}(R)$ as the set of all simplices in $\mathcal{F}$ that have level at most k^* with respect to R. Notice that the simplices in $\mathcal{M}(R)$ do not have to be present in $\mathcal{T}(R)$. Finally, for $R' \subseteq R \subseteq H$ we define $\mathcal{L}(R, R') := \mathcal{T}(R) \cap \mathcal{M}(R')$. The idea is that the set $\mathcal{L}(R, R')$ corresponds to the structure maintained by the lazy algorithm: in our analysis R will be the current subset and R' will be the subset for which the latest clean-up was performed. Observe that $\mathcal{L}(R, R)$ is the same as $\mathcal{L}_{k^*}(R)$, the set of simplices of the bottom-vertex triangulation of the $\leq k^*$-level of $\mathcal{A}(R)$.

Let $b(\Delta) := |\mathcal{D}(\Delta)|$ and $\omega(\Delta) := |\mathcal{K}(\Delta)|$. It is straightforward to verify that the following three conditions hold:

(i) there is a constant $b > 0$ depending on d such that $b(\Delta) \leq b$ for all $\Delta \in \mathcal{F}$,

(ii) a region $\Delta \in \mathcal{F}$ is in $\mathcal{T}(R)$ if and only if $\mathcal{D}(\Delta) \subseteq R$ and $\mathcal{K}(\Delta) \cap R = \emptyset$,

(iii) for $R' \subseteq R \subseteq H$ we have $\mathcal{M}(R) \subseteq \mathcal{M}(R')$.

Let $h_1, \ldots, h_n$ be a random permutation of H, and let $H_r := \{h_1, \ldots, h_r\}$. Define $\mathcal{L}_r := \mathcal{L}(H_r, H_p)$. (Recall that p is the largest power of two that is less than r.) Furthermore, for $1 \le q < r \le n$, we define the function

$$\tau(r,q) := \max_{q<t\le r} E\big[|\mathcal{L}(H_t, H_q)|\big]. \qquad (1)$$

De Berg et al. [dBDS94] proved that under conditions *(i)–(iii)* the following theorem holds.

Theorem 1 *Let $3 \le r \le n$ and let p be the largest power of two that is less than r. The expected size of $\mathcal{L}_r$ is at most $\tau(r,p)$. The expected number of simplices in $\mathcal{L}_r \setminus \mathcal{L}_{r-1}$ is bounded by $O(\frac{1}{r}\tau(r,p/2))$. The expected total conflict size of these simplices is bounded as*

$$E\big[\sum\nolimits_{\Delta\in\mathcal{L}_r\setminus\mathcal{L}_{r-1}} \omega(\Delta)\big] \le O(\frac{n}{r^2}\tau(r,p/2)).$$

To apply this theorem we must bound the function $\tau(r,q)$ for $r/4 \le q < r$. In other words, we must bound the expected number of simplices in the bottom-vertex triangulation $\mathcal{T}(H_t)$ that have level at most k^* with respect to H_q, for $r/4 \le q < t \le r$. It is sufficient to bound the expected number of vertices of $\mathcal{A}(H_t)$ at level at most k^* with respect to H_q; the number of simplices is linear in this number. Observe that H_q is a random sample of H_t. We shall reformulate our problem in an abstract setting so that we can use results on random sampling.

Recall that $\mathcal{L}(H_q, H_q)$ equals $\mathcal{L}_{k^*}(H_q)$, the set of simplices of the bottom-vertex triangulation of the $\le k^*$-level of $\mathcal{A}(H_q)$. As before, let $\mathcal{F}$ denote the set of all possible simplices, and let $\mathcal{D}(\Delta)$ denote the defining set of a simplex Δ. Again, the following condition holds:

(i') there is a constant $b > 0$ such that $b(\Delta) \le b$ for all $\Delta \in \mathcal{F}$,

Let $\mathcal{K}(\Delta, H_t)$ be the subset of hyperplanes in H_t that intersect the interior of Δ. It is not difficult to check that the following conditions hold for every subset $R \subseteq H_t$:

(ii') if $\Delta \in \mathcal{L}(R,R)$, then $\mathcal{D}(\Delta) \subseteq R$,

(iii') if $\Delta \in \mathcal{L}(R,R)$, then $\mathcal{K}(\Delta, H_t) \cap R = \emptyset$,

(iv') if $\Delta \in \mathcal{L}(R,R)$ and R' is a subset of R with $\mathcal{D}(\Delta) \subseteq R' \subseteq R$, then $\Delta \in \mathcal{L}(R', R')$.

Define $\omega(\Delta, H_t) := |\mathcal{K}(\Delta, H_t)|$. The number of vertices we have to consider is

$$O(\sum_{\Delta\in\mathcal{L}_{k^*}(H_q)} (1 + \omega(\Delta, H_t))^d).$$

To bound the expression above we now use the fact that H_q is a random sample of H_t of size q with $q \ge t/4$. Because conditions *(i')–(iv')* hold we can use a theorem by de Berg et al. [dBDS94] for bounding higher moments. This theorem tells us that

$$E\big[\sum_{\Delta\in\mathcal{L}_{k^*}(H_q)} \omega(\Delta, H_t)^d\big] = O\big((t/q)^d |\mathcal{L}_{k^*}(H_q)|\big).$$

The complexity of $\mathcal{L}_{k^*}(H_q)$, the $\le k^*$-level of $\mathcal{A}(H_q)$, is $O(q^{\lfloor d/2 \rfloor}(k^*)^{\lceil d/2 \rceil})$. We can conclude that $\tau(r,q) = O(r^{\lfloor d/2 \rfloor}(k^*)^{\lceil d/2 \rceil})$ for $r/4 \le q < r$. Combining this result with Theorem 1 gives us the following result. (To simplify the expression we have replaced occurrences of p and $p/2$ by r; this does not influence our bounds asymptotically because $r/2 \le p < r$.)

Lemma 2 *The expected running time of the lazy randomized algorithm is*

$$O\big(n \sum_{r=1}^{n} r^{\lfloor d/2 \rfloor - 2}(\kappa(r) + 1)^{\lceil d/2 \rceil}\big).$$

For dimensions larger than three, we simply choose $\kappa(r) := k$. In this case, a cell that we discard cannot contain any part of the $\le k$-level later on, so our algorithm always succeeds. We obtain an algorithm for the computation of $\le k$-levels with expected running time $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$. The algorithm is nearly identical to Mulmuley's algorithm and has the same optimal running time; the only difference is that we replaced his peeling steps by our clean-up steps.

For dimensions two and three, however, this choice of $\kappa(r)$ gives us an expected running time of $O(nk^{\lceil d/2 \rceil} \log(n/k))$, which is suboptimal by a factor of $\log(n/k)$. We now show that a slight twist in the algorithm, namely a different choice of $\kappa(r)$, improves its expected running time for sufficiently large values of k.

In the following, the dimension d is 2 or 3. To minimize the running time we would like to choose $\kappa(r)$ as small as possible. As remarked earlier, however, there is a slight possibility that the algorithm does not give us the entire $\le k$-level, that is, that we have discarded cells that contain a vertex of the $\le k$-level. We must choose $\kappa(r)$ large enough so that the probability of this happening is small. To bound this probability we need the following lemma, which can be proven using tail estimates for the binomial distribution [AS93].

Lemma 3 *Let v be a vertex in an arrangement $\mathcal{A}(H)$ of n hyperplanes at level $\ell_H(v) \le k$, and let $r < n$ and $\beta > 1$ be parameters. The probability that for* *a random sample $R \subseteq H$ of r hyperplanes the level $\ell_R(v)$ of v is more than $2\beta k\frac{r}{n}$ is bounded by*

$$2[e^{\beta-1}\beta^{-\beta}]^{2k(r/n)}.$$

For $\beta \geq e^2$, this is bounded by

$$2\exp(-2\beta k(r/n)).$$

In the remainder we shall need that the probability that a vertex v at level at most k in $\mathcal{A}(H)$ has a level greater than $\kappa(r)$ in $\mathcal{A}(H_r)$ is bounded by $2/n^4$. From the lemma above it readily follows that this is true if we choose

$$\kappa(r) \ := \ \max(2e^2k\frac{r}{n}, 4\ln n).$$

Now let us consider the probability that the algorithm fails. Our choice of $\kappa(r)$ implies that the probability that a vertex v is mistakenly discarded in any given clean-up step is at most $2/n^4$. There are $\lfloor \log n \rfloor$ clean-up phases, and there are at most n^d vertices to consider (with $d = 2$ or $d = 3$), so the probability that we ever discard a vertex unjustified is at most $2\log n/n$. In other words, we expect to succeed after we have run the algorithm a constant number of times.

According to Lemma 2 the expected running time of the algorithm with the given choice of $\kappa(r)$ will be

$$\begin{aligned} & O(n)\sum_{r=1}^{n}\frac{(\kappa(r)+1)^{\lceil d/2\rceil}}{r} \\ \leq\ & O(n)\sum_{r=1}^{n}\Big(\frac{k^{\lceil d/2\rceil}r^{\lceil d/2\rceil-1}}{n^{\lceil d/2\rceil}}+\frac{(\ln n)^{\lceil d/2\rceil}}{r}\Big) \\ =\ & O(nk^{\lceil d/2\rceil}+n(\log n)^{\lceil d/2\rceil+1}). \end{aligned}$$

For $k < 4\ln n$, this is no improvement over the trivial choice of $\kappa(r)$. Hence, our final choice for $\kappa(r)$ is

$$\kappa(r) \ := \ \begin{cases} k, & \text{if } k < 4\ln n, \\ \max(2e^2k\frac{r}{n}, 4\ln n), & \text{otherwise.} \end{cases}$$

This gives us an expected running time of $O(nk^{\lceil d/2\rceil} + n\log n\min(k^{\lceil d/2\rceil}, \log^{\lceil d/2\rceil} n))$, which is optimal for $k \geq \log n$ in the planar case, and for $k \geq \log^{3/2} n$ in the three-dimensional case.

Before we state our theorem on computing the $\leq k$-level we discuss a generalization of the planar algorithm.

Let H be a set of n x-monotone Jordan curves in the plane with each pair of curves intersecting at most s times. To apply our algorithm to arrangements of Jordan curves we need a technique for decomposing such arrangements into constant complexity pieces, or *boxes.* We use the vertical decomposition: we extend a vertical segment upward from every intersection point of two curves until it hits another curve and extend a vertical segment downward until it hits another curve; if a vertical segment does not intersect any curve, it is extended to infinity. The same technique can be applied to the cells in the $\leq\kappa(r)$-level of the arrangement $\mathcal{A}(H_r)$. The number of resulting boxes is $O(\kappa(r)^2\lambda_s(r/\kappa(r)))$. We can now use the lazy randomized incremental algorithm described above to compute the $\leq k$-level in an arrangement of Jordan curves. We only need to change the way the decomposition is updated after an insertion, since we are now using vertical decompositions instead of bottom-vertex triangulations. We omit the details of how the vertical decomposition is updated; they are the same as in randomized incremental algorithms for computing full planar arrangements [Mul91a].

The analysis of the algorithm follows the analysis given earlier, with the maximum complexity of the $\leq k$-level in a set of hyperplanes replaced by the maximum complexity of the $\leq k$-level in a set of Jordan curves. Thus, if p is the largest power of two less than r then the expected running time is

$$O(\sum_{r=1}^{n}\frac{n}{r^2}\tau(r, p/2)),$$

where $\tau(r, p/2) = O(\kappa(r)^2\lambda_s(r/\kappa(r)))$. The same choice of $\kappa(r)$ as before gives us a running time of $O(k^2\lambda_s(n/k) + \min(\lambda_s(n)\log^2 n, k^2\lambda_s(n/k)\log n))$. This bound subsumes the case of line arrangements, since two lines intersect at most once and $\lambda_1(n) = n$.

The same algorithm also works for a set of discs. Using the fact that the complexity of the $\leq k$-level of a set of n discs is $O(nk)$, it can shown that the expected running time of the algorithm, as in the case of lines, is $O(nk + \min(n\log^2 n, nk\log n))$. We omit the easy details from here.

Theorem 4
(i) The $\leq k$-level of an arrangement of n x-monotone Jordan curves in the plane such that any pair intersects in at most s points can be computed in $O(k^2\lambda_s(n/k) + \min(\lambda_s(n)\log^2 n, k^2\lambda_s(n/k)\log n))$ time by a lazy randomized incremental algorithm.
(ii) The $\leq k$-level of an arrangement of n planes in three-dimensional space can be computed in expected time $O(nk^2 + \min(n\log^3 n, nk^2\log n))$ by a lazy randomized incremental algorithm.
(iii) The $\leq k$-level of an arrangement of n discs in the plane can be computed in expected time $O(nk + \min(n\log^2 n, nk\log n))$.

3 Computing order-k Voronoi diagrams

Let P be a set of n points in the plane. The order-k Voronoi diagram of P is the subdivision of the plane into cells such that the k closest points in P are uniquely determined inside each cell. In other words, two points p, q are in the same cell if the k points of P that are closest to p are also the k points of P that are closest to q. P can be mapped to a set $H = H(P)$ of planes in three-dimensional space as follows. Lift each point of P to the unit paraboloid $U : z = x^2 + y^2$, and take the plane that is tangent to U at that point. The order-k Voronoi diagram of P corresponds to the k-level of H—see for example Edelsbrunner's book [Ede87]. The maximum complexity of the order-k Voronoi diagram and, hence, the maximum complexity of the k-level in the situation above, is $\Theta(k(n-k))$.

We give a simple algorithm that computes the k-level for this case in $O(k(n-k)\log n + n\log^3 n)$ time. The algorithm is randomized incremental, like the one in the previous section. This time, however, we do not use the lazy paradigm: we clean up the structure after every step. Another difference with the previous algorithm is the following.

In the algorithm for computing the $\leq k$-level we observed that at time r the $\leq k$-level of the full set is expected to be contained in the $\leq kr/n$-level of the sample. Stated differently, we made an estimation of the real level of a vertex in the arrangement of sample planes based on its level in the sample. We then decided whether or not to discard the vertex based on this estimation. By making our estimations conservatively we ensured that the probability of discarding a "good" vertex was small.

This line of attack turns out not to work for computing the exact k-level. So we maintain a portion of the arrangement which is guaranteed to contain the k-level. In particular, we do not make an estimate of the real level of a vertex based on its level in the sample. Instead we maintain for each vertex in the current arrangement a range of values that is guaranteed to contain its real level. We only discard a vertex when k is not in this range. Thus we are sure that the algorithm produces the right answer. To bound the expected running time of our algorithm, however, we have to use a probabilistic argument.

As before, we denote by H the set of planes and by H_r the first r elements in a random permutation of H. Let Δ be a simplex in the bottom-vertex triangulation of the arrangement $\mathcal{A}(H_r)$. The *conflict list* $H(\Delta)$ of Δ is the set of planes in H that intersect the interior of Δ. Let $\ell(\Delta)$ be the level of any point in the interior of Δ with respect to $H \setminus H(\Delta)$ and let $\omega(\Delta) := |H(\Delta)|$. We call Δ *interesting* if $\ell(\Delta) \leq k \leq \ell(\Delta) + \omega(\Delta)$, and we call a cell of $\mathcal{A}(H_r)$ *active* if its bottom-vertex triangulation has at least one interesting simplex.

Our algorithm maintains the simplices of the bottom-vertex triangulation of all active cells of $\mathcal{A}(H_r)$. Since the k-level of $\mathcal{A}(H)$ can intersect only interesting simplices, we are sure that the k-level is contained in the region that we maintain. With every simplex Δ we maintain the conflict list $H(\Delta)$ and the level $\ell(\Delta)$. We also store with every vertex of each active cell its real level, that is, its level with respect to H. This will help us to compute the level $\ell(\Delta)$ when we create a new simplex Δ. Finally, we maintain the adjacency graph on the simplices; there is an arc between two simplices if they share a facet.

To add a new plane h_{r+1} we proceed as follows. First we identify all active cells intersected by h_{r+1}; using the conflict lists this can be done in time linear in the number of simplices intersected by h_{r+1}. Then we re-triangulate these cells and we compute the conflict lists for the new simplices, as described in the previous section. Every new simplex Δ contains at least one old vertex; using the level of this vertex we can compute the levels of the new vertices of Δ by examining the planes in $H(\Delta)$. Hence, we can compute the real levels of all new vertices in time that is linear in the size of the new conflict lists. Next we compute the level $\ell(\Delta)$ of each new simplex Δ; using the level of any of the vertices of Δ this can again be done by examining the planes in $H(\Delta)$. (Note that $\ell(\Delta)$ does not change for the old simplices.) Finally, we check which of the new simplices is interesting, and we discard the cells that have no interesting simplex. These two steps can be done in time linear in the total size of the conflict lists of all deleted simplices.

When the last hyperplane h_n has been added, we have $H(\Delta) = \emptyset$ for every Δ. In other words, $\omega(\Delta) = 0$ and we are left with the simplices of the bottom-vertex triangulation of $\mathcal{A}(H)$ that have level k. The facets of these simplices that bound their simplex from below and are contained in one of the planes in H form the k-level of $\mathcal{A}(H)$.

It remains to analyze the algorithm.

Lemma 5 *The expected number of simplices present at time r is $O(rk\frac{r}{n}\log r + r\log^2 r)$.*

Proof: We use an argument based on the ε-net theory. Haussler and Welzl [HW87] proved that a random sample R of r planes from a set H of n planes has the following property with probability at least

$1-1/n^3$: Any line segment s, which do not intersect a plane in R, intersects at most $C\frac{n}{r}\log r$ planes in H, where C is a constant.

The set H_r is a random sample of H of size r. Let us therefore assume for a moment that the above mentioned property holds for H_r, that is, every segment s that lies within a cell of $\mathcal{A}(H_r)$ intersects at most $C\frac{n}{r}\log r$ planes of H. As a consequence, for all simplices Δ present in stage r we have $\omega(\Delta) \leq 3C\frac{n}{r}\log r$. We now bound the total complexity of all active cells (still assuming that the property above holds for H_r). By definition, every active cell $\mathcal{C}$ contains at least one interesting simplex Δ. Let p be an arbitrary point inside Δ. The level of p with respect to H lies between $\ell(\Delta)$ and $\ell(\Delta)+\omega(\Delta)$. Because Δ is interesting and $\omega(\Delta) \leq 3C\frac{n}{r}\log r$ we know that the level of p lies between $k-3C\frac{n}{r}\log r$ and $k+3C\frac{n}{r}\log r$. We now observe that every vertex v of $\mathcal{C}$ can be connected to p with a segment $s = \overline{pv}$ lying inside $\mathcal{C}$. Therefore s crosses no more than $C\frac{n}{r}\log r$ planes in H, and we can conclude that the level of v lies between $k-4C\frac{n}{r}\log r$ and $k+4C\frac{n}{r}\log r$. The number of vertices in $\mathcal{A}(H)$ with this level is bounded by

$$\sum_{\ell=k-4C\frac{n}{r}\log r}^{k+4C\frac{n}{r}\log r} O(n\ell) = O(nk\frac{n}{r}\log r + n(\frac{n}{r}\log r)^2).$$

The probability that a given vertex of $\mathcal{A}(H)$ appears in $\mathcal{A}(H_r)$ is $\Theta((\frac{r}{n})^3)$. Hence, the total complexity of all active cells (and thus also the number of simplices present) at time r is $O(rk\frac{r}{n}\log r + r\log^2 r)$.

This derivation was under the assumption that the ε-net property mentioned above holds for H_r. This is true with probability at least $1-1/n^3$. If the property is not true for H_r, we use the trivial bound of $O(r^3)$ on the number of simplices in $\mathcal{A}(H_r)$. The expected number of simplices is therefore bounded by

$$\begin{aligned}&(1-1/n^3)\cdot O(rk\tfrac{r}{n}\log r + r\log^2 r) \ + \ (1/n^3)\cdot O(r^3)\\ =\ & O(rk\tfrac{r}{n}\log r + r\log^2 r).\end{aligned}$$ □

As in the previous section, the expected running time of the algorithm is bounded by the sum of the sizes of the conflict lists of all simplices ever created during the algorithm. Using Theorem 1 again (with the appropriate definitions of $\mathcal{L}$ and τ), we can prove that this quantity is bounded by

$$\sum_{r=1}^{n} \frac{n}{r^2} O(rk\frac{r}{n}\log r + r\log^2 r) = O(nk\log n + n\log^3 n).$$

For $k \leq n/2$ this is equal to $O(k(n-k)\log n + n\log^3 n)$. For $k \geq n/2$, we consider the k-level as the $(n-k)$-level in the arrangement turned upside down, leading to the following theorem.

Theorem 6 *The k-level in an arrangement of n planes in three-dimensional space that are all tangent to the unit paraboloid can be computed in expected time $O(k(n-k)\log n + n\log^3 n)$ with a randomized incremental algorithm, using $O(k(n-k)\log n + n\log^2 n)$ storage.*

Remark. We only used the fact that all planes in the set H are tangent to the unit paraboloid in the analysis of the algorithm; the algorithm itself works for arbitrary sets of planes. Hence, if the complexity of the k-level for arbitrary sets of planes is about the same as in the restricted case of planes tangent to the unit paraboloid, then the running time of our algorithm is also close to optimal in this general case.

Corollary 7 *The order-k Voronoi diagram of n points in the plane can be computed in expected time $O(k(n-k)\log n + n\log^3 n)$ with a randomized incremental algorithm that uses $O(k(n-k)\log n + n\log^2 n)$ storage.*

4 Concluding Remarks

We have presented algorithms concerning levels in arrangements. The first algorithm computes the $\leq k$-level in an arrangement of curves in the plane or planes in three-dimensional space. The second algorithm computes the exact k-level of a set of planes in three-dimensional space that are tangent to the unit paraboloid; the k-level in this situation corresponds to the order-k Voronoi diagram of a set of points in the plane. The algorithms improve and/or simplify the best known algorithms for these problems.

Both algorithms are randomized and incremental. We improve upon previous randomized incremental algorithms for these problems by being more careful in what to maintain. In particular, previous randomized incremental algorithms for these problems maintained a part of the current arrangement based on the level of that part in the current arrangement. The part that we maintain, on the other hand, is based on the level of that part in the full arrangement. This level is, of course, not known exactly during the algorithm. For the computation of the $\leq k$-level we therefore made an approximation of the level in the full arrangement based on the level in the current arrangement. For the computation of the exact k-level we maintained a range of values that is guaranteed to contain the level in the full arrangement.

Our algorithms are suboptimal when k is very small. We leave it as an open problem to develop an algorithm that is optimal for all values of k. Another open problem, even in the plane, is to compute the exact k-level in optimal time.

References

[AS92] F. Aurenhammer and O. Schwarzkopf. A simple on-line randomized incremental algorithm for computing higher order Voronoi diagrams. *Internat. J. Comput. Geom. Appl.*, 2:363–381, 1992.

[AS93] N. Alon and J. Spencer. *The Probabilistic Method.* J. Wiley & Sons, 1993.

[ASS89] P. K. Agarwal, M. Sharir, and P. Shor. Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences. *J. Combin. Theory Ser. A*, 52:228–274, 1989.

[BDT93] J.-D. Boissonnat, O. Devillers, and M. Teillaud. An semidynamic construction of higher-order Voronoi diagrams and its randomized analysis. *Algorithmica*, 9:329–356, 1993.

[CE87] B. Chazelle and H. Edelsbrunner. An improved algorithm for constructing kth-order Voronoi diagrams. *IEEE Trans. Comput.*, C-36:1349–1354, 1987.

[Cla87] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 2:195–222, 1987.

[Cla88] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17:830–847, 1988.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[CSY87] R. Cole, M. Sharir, and C. K. Yap. On k-hulls and related problems. *SIAM J. Comput.*, 16:61–77, 1987.

[dBDS94] M. de Berg, K. Dobrindt, and O. Schwarzkopf. On lazy randomized incremental construction. In *Proc. 25th Annu. ACM Sympos. Theory Comput.*, 1994.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Heidelberg, West Germany, 1987.

[ERvK93] H. Everett, J.-M. Robert, and M. van Kreveld. An optimal algorithm for the ($\leq k$)-levels, with applications to separation and transversal problems. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 38–46, 1993.

[HW87] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[Mul91a] K. Mulmuley. A fast planar partition algorithm, II. *J. ACM*, 38:74–103, 1991.

[Mul91b] K. Mulmuley. On levels in arrangements and Voronoi diagrams. *Discrete Comput. Geom.*, 6:307–338, 1991.

[Mul93] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, New York, 1993.

[PSS92] J. Pach, W. Steiger, and E. Szemerédi. An upper bound on the number of planar k-sets. *Discrete Comput. Geom.*, 7:109–123, 1992.

[Sha91] M. Sharir. On k-sets in arrangements of curves and surfaces. *Discrete Comput. Geom.*, 6:593–613, 1991.

Computing Many Faces in Arrangements of Lines and Segments *

Pankaj K. Agarwal[†] Jiří Matoušek[‡] Otfried Schwarzkopf[§]

Abstract

We present randomized algorithms for computing many faces in an arrangement of lines or of segments in the plane, which are considerably simpler and slightly faster than the previously known ones. The main new idea is a simple randomized $O(n \log n)$ expected time algorithm for computing $\sqrt{n}$ cells in an arrangement of n lines.

1 Introduction

Given a finite set of lines, L, in the plane, the *arrangement* of L, denoted as $\mathcal{A}(L)$, is the cell complex induced by L. The 0-faces (or *vertices*) of $\mathcal{A}(L)$ are the intersection points of L, the 1-face (or *edges*) are maximal portions of lines of L that do not contain any vertex, and the 2-faces (called *cells*) are the connected components of $\mathbb{R}^2 - \bigcup L$. For a finite set S of segments we define the arrangement, $\mathcal{A}(S)$, in an analogous manner. Notice that while the cells in a line arrangement are convex, in an arrangement of segments they need not even be simply connected.

Line and segment arrangements have been extensively studied in computational geometry (as well as in some other areas), as a wide variety of computational geometry problems can be formulated in terms of computing such arrangements or their parts [11, 14].

Given a set L of n lines and a set P of m points in the plane, we define $\mathcal{A}(L, P)$ to be the collection of all cells of $\mathcal{A}(L)$ containing at least one point of P. The *combinatorial complexity* of a cell C, denoted by $|C|$, in $\mathcal{A}(L)$ is the number of edges of C. Let $\kappa(L, P) = \sum_{C \in \mathcal{A}(L,P)} |C|$ denote the total combinatorial complexity of all cells in $\mathcal{A}(L, P)$, and let

$$\kappa(n, m) = \max \kappa(L, P)\,,$$

where the maximum is taken over all sets of n lines and over all sets of m points in the plane. It is known that

$$\kappa(n, m) = \Theta(n^{2/3} m^{2/3} + n + m)\,.$$

The upper bound was proven by Clarkson et al. [9]; previous results and related work can be found in Canham [4], Edelsbrunner and Welzl [13], Szemerédi and Trotter [20]

In this paper we study the problem of computing $\mathcal{A}(L, P)$, that is, for each cell $C \in \mathcal{A}(L, P)$, we want return the vertices of C in, say, clockwise order. We will refer to the cells of $\mathcal{A}(L, P)$ as the marked cells of $\mathcal{A}(L)$. Edelsbrunner et al. [12] presented a randomized algorithm, based on the random sampling technique [16], for computing $\mathcal{A}(L, P)$, whose expected running time was

$$O(m^{2/3-\varepsilon} n^{2/3+2\varepsilon} \log n + n \log n \log m)\,,$$

for any fixed $\varepsilon > 0$. A deterministic algorithm with running time

$$O(m^{2/3} n^{2/3} \log^{O(1)} n + n \log^3 n + m \log n)$$

*A part of this work was done while the first and third authors were visiting Charles University and while the first author was visiting Utrecht University. The first author has been supported by National Science Foundation Grant CCR–93–01259 and an NYI award. The second author has been supported by Charles University grant No. 351 and Czech Republic Grant GAČR 201/93/2167. The third author has been supported by the Netherlands' Organization for Scientific Research (NWO) and partially supported by ESPRIT Basic Research Action No. 7141 (project ALCOM II: *Algorithms and Complexity*)

[†] Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129, USA.

[‡] Department of Applied Mathematics, Charles University, Malostranské nám. 25, 118 00 Praha 1, Czech Republic.

[§] Department of Computer Science, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

was given by Agarwal [1]. These algorithms thus are nearly worst-case optimal, but both of them are rather involved.

Recently randomized incremental algorithms have been developed for a wide variety of geometric problems, which add the input objects one by one in a random order and maintain the desired structure; see e.g. [6, 10, 18, 19]. In our case, we can add the lines of L one by one in a random order and maintain the marked cells in the arrangement of lines added so far. However, this approach seems to yield expected running time of $\Omega(n\sqrt{m}+m\log n)$ in the worst case. We, therefore, do not quite follow the randomized incremental paradigm.

We begin by presenting an expected $O(m^2+n\log n)$ time randomized algorithm for computing $\mathcal{A}(L,P)$. Notice that for $m \le \sqrt{n\log n}$, this algorithm is optimal. We then apply the random sampling technique in a standard way, obtaining an expected $O(m^{2/3}n^{2/3}\log^{2/3}\frac{n}{\sqrt{m}}+(m+n)\log n)$ time algorithm.

We also study a similar but more complicated problem of computing the marked cells in an arrangement of n segments. Let S be a set of n segments in the plane. We use an analogous notation $\mathcal{A}(S,P)$ to denote the set of the cells in $\mathcal{A}(S)$ containing at least one point of P, and $\eta(n,m)$ to denote the maximum combinatorial complexity of $\mathcal{A}(S,P)$ over all sets S of n segments and sets P of m points in the plane. Aronov et al. [2] proved that

$$\eta(n,m) = O\left(m^{2/3}n^{2/3} + n\log m + n\,\alpha(n)\right).$$

A randomized algorithm with expected running time $O(m^{2/3-\varepsilon}n^{2/3+2\varepsilon}\log n + n\,\alpha(n)\log^2 n\log m)$ is described by Edelsbrunner et al. [12], and a slightly faster deterministic algorithm is presented by Agarwal [1].

Following a similar strategy as for the case of lines, we first develop a randomized algorithm with $O((m^2 + n\log m + n\,\alpha(n))\log n)$ expected running time. Let's remark that the above upper bound for $\eta(n,m)$ is not known to be tight, and a bound like $\eta(n,\sqrt{n}) = O(n\,\alpha(n))$ (which is conjectured to be the complexity of $\sqrt{n}$ cells) will immediately improve the expected running time of our algorithm to $O(n\log n\,\alpha(n))$. Plugging this algorithm to the standard random sampling technique, as in the case of lines, we obtain a randomized algorithm for computing $\mathcal{A}(S,P)$ in $O(m^{2/3}n^{2/3}\log^{4/3}\frac{n}{\sqrt{m}}\,\alpha^{1/3}(\frac{n}{\sqrt{m}}) + (m + n\log m + n\,\alpha(n))\log n)$ expected time. If the segments of S have only $k = o(n^2)$ intersection points, the expected running time of the algorithm is $O(m^{2/3}k^{1/3}\log^{4/3}\frac{k}{m}\alpha^{1/3}(\frac{k}{m})+(m+n\log m+n\,\alpha(n))\log n)$.

For the analysis of the expected running time of our algorithms we will use a generalization of a lemma due to Chazelle and Friedman [7]. (An alternative analysis could probably be obtained using a method similar to that of Chazelle et al. [6], but we hope that our approach is somewhat more intuitive).

2 A generalization of Chazelle and Friedman's lemma

Let S be a set of lines or segments, and P a set of points in the plane. For a cell C of the collection $\mathcal{A}(S,P)$, let $C^{||}$ denote the collection of trapezoids in the vertical decomposition of C,[1] and let $\mathcal{A}^{||}(S,P) = \bigcup_{C\in\mathcal{A}(S,P)} C^{||}$ denote the set of trapezoids in the vertical decomposition of $\mathcal{A}(S,P)$. Abusing the notation slightly, we will use $\mathcal{A}^{||}(S,P)$ to denote the corresponding planar subdivision as well.

Let R be a subset of S. For a trapezoid $\Delta \in \mathcal{A}^{||}(R,P)$, let $w(\Delta)$ denote the number of elements of S intersecting the interior of Δ.

Let $n = |S|$, and suppose R is a random subset of S of size r. For the analysis of our algorithms, we are interested in estimating the expectation, over all random choices of R,

$$\mathrm{E}\Big[\sum_{\Delta\in\mathcal{A}^{||}(R,P)} w(\Delta)^c\Big], \tag{2.1}$$

where c is a small constant like $c = 2$. Well-known results concerning the so-called ε-nets (Haussler and Welzl [16]) imply that, for every $\Delta \in \mathcal{A}^{||}(R)$, $w(\Delta) \le C(n/r)\log r$ with high probability, where C is a suitable constant. From this one can derive a bound for (2.1). We are, however, interested in the following, slightly stronger bound (better by a factor of $\log^c r$):

Proposition 2.1 *(i) Let L be a set of n lines and P a set of m points in the plane. If $R \subseteq L$ is random subset of size r, where each subset of size r is chosen with equal probability, then for any constant $c \ge 1$.*

$$\mathrm{E}\,[\sum_{\Delta\in\mathcal{A}^{||}(R,P)} w(\Delta)^c] = \kappa(r,m)\cdot O((n/r)^c)\,.$$

(ii) Let S be a set of n segments and P a set of m points in the plane. If $R \subseteq S$ is random subset

[1] The vertical decomposition $C^{||}$ of a cell C in an arrangements of segments (or of lines) is obtained by drawing a vertical line from each vertex of C in both directions (within C) until it hits another edge of C.

of size r, where each subset of size r is chosen with equal probability, then for any constant $c \geq 1$.

$$\mathrm{E}\,[\sum_{\Delta \in \mathcal{A}^{||}(R,P)} w(\Delta)^c] = \eta(r,m) \cdot O((n/r)^c)\,.$$

These bounds essentially say that the cth moment of the quantities $w(\Delta)$ behaves as if they were $O(n/r)$. If we sum $w(\Delta)$ over all cells in $\mathcal{A}(R)$—the case where every cell of $\mathcal{A}(R)$ contains a point of P—then Proposition 2.1 follows from a result of Clarkson and Shor [10]. In our situation, where the sum is taken over only some of the cells, the Clarkson-Shor framework does not apply directly anymore (the main distinction between these two situations will be outlined below). We give a proof based on a generalization of an approach due to Chazelle and Friedman [7], which is somewhat different from the Clarkson-Shor method. Recently, de Berg et al. [3] gave an alternative proof of Proposition 2.1.

We derive a key lemma in a somewhat abstract framework; see also [6, 7, 10] for various approaches to axiomatize similar situations.

Let S be a set of objects. For a subset $R \subseteq S$, we define a collection of 'regions' called $CT(R)$; in the situation of Proposition 2.1 the objects are segments, the regions are trapezoids and $CT(R) = \mathcal{A}^{||}(R,P)$. Let $T = T(S) = \bigcup_{R \subseteq S} CT(R)$ denote the set of regions defined by all possible subsets of S. We associate two subsets $D(\Delta), K(\Delta) \subseteq S$ with each region $\Delta \in T$.

$D(\Delta)$, called the *defining set*, is a subset of S defining the region Δ in a suitable geometric sense.[2] We assume that for every $\Delta \in T$, $|D(\Delta)| \leq d$ for a (small) constant d. In Proposition 2.1, each trapezoid Δ is defined by at most 4 segments (or lines) of S, which constitute the set $D(\Delta)$; details can be found in Chazelle et al. [6].

$K(\Delta)$, called the *killing set*, is a set of objects of S, such that including any object of $K(\Delta)$ into R prevents Δ from appearing in $CT(R)$. In many applications $K(\Delta)$ is the set of objects intersecting the cell Δ; this is also the case in Proposition 2.1. Set $w(\Delta) = |K(\Delta)|$.

Let $S, CT(R), D(\Delta), K(\Delta)$ be such that for any subset $R \subseteq S$, $CT(R)$ satisfies the following axioms:

(i) For any $\Delta \in CT(R)$, $D(\Delta) \subseteq R$ and $R \cap K(\Delta) = \emptyset$, and

(ii) If $\Delta \in CT(R)$ and R' is a subset of R with $D(\Delta) \subseteq R'$, then $\Delta \in CT(R')$.

It is easily checked that these axioms hold in the situations of Proposition 2.1.

For any natural number t, let us denote

$$CT_t(R) = \{\Delta \in CT(R) \mid w(\Delta) \geq tn/r\}\,.$$

We establish the following:

Lemma 2.2 *Given a set S of objects, let R be a random sample of size $r \leq n = |S|$ drawn from S, and let t be a parameter, $1 \leq t \leq r/d$, where $d = \max |D(\Delta)|$. Assuming that $CT(R)$, $D(\Delta)$ and $K(\Delta)$ satisfy Axioms (i) and (ii) above, we have*

$$\mathrm{E}\,|CT_t(R)| = O(2^{-t}) \cdot \mathrm{E}\,|CT(R')|\,, \qquad (2.2)$$

where $R' \subseteq S$ denotes a random sample of size $r' = \lfloor r/t \rfloor$.

Roughly speaking, Lemma 2.2 says that the expected number of "large" trapezoids in $CT(R)$, that is trapezoids which the value of $w(\Delta)$ exceeds the "right" value n/r more than t times, decreases exponentially with t.

Chazelle and Friedman [7] proved a result analogous to Lemma 2.2 under the following stronger axiom replacing (ii):

(ii′) If $D(\Delta) \subseteq R$ and $K(\Delta) \cap R = \emptyset$, then $\Delta \in CT(R)$.

This assumption implies that the presence of Δ in $CT(R)$ depends only on $D(\Delta)$ and $K(\Delta)$, thus it is determined purely "locally." Notice that (ii′) may fail in the situation of Proposition 2.1. However, (ii′) holds in the special case, when $CT(R)$ is the vertical decomposition of *all* cells in $\mathcal{A}(R)$.

Proof of Lemma 2.2: Let $T_t = \bigcup_{R \subseteq S} CT_t(R)$. We have

$$\begin{aligned} \mathrm{E}\,|CT_t(R)| &= \sum_{\Delta \in T_t} \Pr[\Delta \in CT(R)]\,, \qquad (2.3) \\ \mathrm{E}\,|CT(R')| &= \sum_{\Delta \in T} \Pr[\Delta \in CT(R')] \\ &\geq \sum_{\Delta \in T_t} \Pr[\Delta \in CT(R')]\,. \qquad (2.4) \end{aligned}$$

We will prove that, for each $\Delta \in T_t$,

$$\Pr[\Delta \in CT(R)] = O(2^{-t}) \cdot \Pr[\Delta \in CT(R')]\,, \quad (2.5)$$

which in conjunction with (2.3) and (2.4) implies (2.2).

Let A_Δ denote the event that $D(\Delta) \subseteq R$ and $K(\Delta) \cap R = \emptyset$, and let A'_Δ denote the event $D(\Delta) \subseteq R'$ and $K(\Delta) \cap R' = \emptyset$.

[2] We need not make this precise here, as this is only an intuitive meaning of $D(\Delta)$. The analysis depends only on the axioms involving $D(\Delta)$ given below, and these will be satisfied in our specific examples.

We rewrite $\Pr[\Delta \in CT(R)]$ using the definition of conditional probability:

$$\Pr[\Delta \in CT(R)] = \Pr[A_\Delta] \cdot \Pr[\Delta \in CT(R) \mid A_\Delta]$$

and analogously

$$\Pr[\Delta \in CT(R')] = \Pr[A'_\Delta] \cdot \Pr[\Delta \in CT(R') \mid A'_\Delta]\,.$$

We observe that by Axiom (ii), we have

$$\Pr[\Delta \in CT(R) \mid A_\Delta] \le \Pr[\Delta \in CT(R') \mid A'_\Delta]\,. \quad (2.6)$$

Indeed, $\Pr[\Delta \in CT(R') \mid A'_\Delta]$ is the probability that Δ appears in $CT(R')$, where R' is created as follows: Include all elements of $D(\Delta)$, and then choose the remaining $r' - |D(\Delta)|$ elements randomly among the elements of $S \setminus (D(\Delta) \cup K(\Delta))$. We may continue this experiment by choosing R to be R' plus a random subset of $r - r'$ elements of $S \setminus (R' \cup K(\Delta))$. Clearly for such subsets R' and R, $\Pr[\Delta \in CT(R)] \le \Pr[\Delta \in CT(R')]$. Moreover, the subset R selected by this experiment contains $D(\Delta)$ plus $r - |D(\Delta)|$ random elements of $S \setminus (D(\Delta) \cup K(\Delta))$, so $\Pr[\Delta \in CT(R)]$ is the same as the left hand side of (2.6).

Therefore

$$\frac{\Pr[\Delta \in CT(R)]}{\Pr[\Delta \in CT(R')]} \le \frac{\Pr[A_\Delta]}{\Pr[A'_\Delta]}\,.$$

(Note that $r' = \lfloor r/t \rfloor \ge d$, and hence both denominators are nonzero.)

It remains to estimate the latter ratio, which can be done in the same way as by Chazelle and Friedman. Let $d = |D(\Delta)|$, $w = w(\Delta)$, and for two non-negative integers $a \le x$, let $x^{\underline{a}} = x(x-1)\cdots(x-a+1)$. Then

$$\begin{aligned}\frac{\Pr[A_\Delta]}{\Pr[A'_\Delta]} &= \frac{\binom{n-w-d}{r-d}}{\binom{n}{r}} \cdot \frac{\binom{n}{r'}}{\binom{n-w-d}{r'-d}} \\ &= \frac{r^{\underline{d}}}{r'^{\underline{d}}} \cdot \frac{(n-w-r')^{\underline{r}}}{(n-r')^{\underline{r}}}\,.\end{aligned}$$

By our assumption $r' \ge d$, so we obtain

$$\frac{r-i}{r'-i} \le dt \quad \text{for}\, i = 0, 1, \ldots, d-1.$$

Thus, the first factor in the above expression is $O(t^d)$. To bound the second factor, we observe that, for $i = r', r'+1, \ldots, r-1$,

$$\frac{n-w-i}{n-i} = 1 - \frac{w}{n-i} \le 1 - \frac{w}{n} \le \exp(-w/n)\,.$$

Since $w \ge tn/r$, we have $w/n \ge t/r$, and therefore

$$\begin{aligned}\frac{\Pr[A_\Delta]}{\Pr[A'_\Delta]} &\le O(t^d)\exp\left(\frac{-t(r-r')}{r}\right) \\ &= O(t^d)\exp(-(t-1)) = O(2^{-t})\,,\end{aligned}$$

as desired. □

We now prove Proposition 2.1.

Proof of Proposition 2.1: We will only prove the first part, the second part is identical. For any subset $R \subseteq L$ of size r, let $CT(R)$ denote the set of trapezoids in the vertical decomposition of the marked cells of $\mathcal{A}(R)$, i.e., $CT(R) = \mathcal{A}^{||}(R, P)$. Obviously, $|CT(R)| \le \kappa(r, m)$. Now

$$\begin{aligned}&\mathrm{E}\,[\sum_{\Delta \in \mathcal{A}^{||}(R,P)} w(\Delta)^c] \\ &= \mathrm{E}\left[\sum_{t \ge 1} \left(t\frac{n}{r}\right)^c (|CT_t(R)| - |CT_{t-1}(R)|)\right] \\ &\le \left(\frac{n}{r}\right)^c \sum_{t \ge 0} (t+1)^c \cdot \mathrm{E}\,|CT_t(R)| \\ &= \left(\frac{n}{r}\right)^c \sum_{t \ge 0} t^c O(2^{-t}) \cdot \kappa(r/t, m) \\ &\le \kappa(r, m) \left(\frac{n}{r}\right)^c \sum_{t \ge 0} O(t^c \cdot 2^{-t}) \\ &= \kappa(r, m) \cdot O\left(\left(\frac{n}{r}\right)^c\right)\,.\end{aligned}$$

□

3 Computing cells in line arrangements

In this section we describe a randomized algorithm for computing $\mathcal{A}(L, P)$, where L is a set of n lines and P a set of m points in the plane. In fact, our algorithm computes the vertical decomposition $\mathcal{A}^{||}(L, P)$. Each face of $\mathcal{A}^{||}(L, P)$ is a trapezoid, bounded by at most two vertical segments and portions of at most two edges of a cell of $\mathcal{A}^{||}(L, P)$. We first present a randomized algorithm for computing $\mathcal{A}^{||}(L, P)$ with $O(m^2 + n \log n)$ expected time, which is optimal for $m \le \sqrt{n \log n}$. We assume that the points of P are sorted in nondecreasing order of their x-coordinates, and that the lines of L are sorted by their slopes. We first describe the outline of the overall algorithm, and then discuss each of the steps in detail.

1. Let t be some sufficiently large constant. Choose a random subset $R \subseteq L$ of $r = \lfloor n/t \rfloor$ lines.

2. Partition P into $q = \lceil \sqrt{t}\,\rceil$ subsets $P_1, \ldots, P_q$, each of size at most $k = \lfloor m/\sqrt{t} \rfloor$, where

$$\begin{aligned}P_i &= \{p_{(i-1)k+1}, \ldots, p_{ik}\} \quad \text{for}\, i < q, \\ P_q &= \{p_{(q-1)k+1}, \ldots, p_m\}\,.\end{aligned}$$

3. For each $i \leq q$, compute $\mathcal{A}^{||}(R, P_i)$ recursively. If a cell C of $\mathcal{A}(R)$ is computed more than once, retain only one copy of C. (Note that multiple copies of a cell C are computed if C contains the points of more than one P_i's.) Since P is sorted in the x-direction, it is easy to detect multiple copies of a cell. In this way, obtain $\mathcal{A}^{||}(R, P)$.

4. For each line $\ell \in L \setminus R$, compute the cells of $\mathcal{A}(R, P)$ that ℓ intersects.

5. For each trapezoid Δ of $\mathcal{A}^{||}(R, P)$, compute the set $L_\Delta \subseteq L \setminus R$ of lines that intersect the interior of Δ.

6. For each trapezoid $\Delta \in \mathcal{A}^{||}(R, P)$, compute the arrangement of lines of L_Δ within Δ, and the vertical decompositions of its cells. For each cell $C \in \mathcal{A}(R, P)$, perform a graph search on trapezoids of these vertical decomposition to merge appropriate trapezoids and to discard superfluous ones, thus forming the portion of $\mathcal{A}^{||}(S, P)$ within the cell C.

Steps 1–3 are trivial, so we now describe Steps 4–6 in more detail.

Step 4. We want to compute the cells of $\mathcal{A}(R, P)$ intersected by each line in $L \setminus R$. The situation can be viewed as follows: we have a collection $\mathcal{C}$ of disjoint convex polygons (the cells of $\mathcal{A}(R, P)$), and a set $L \setminus R$ of lines. The collection $\mathcal{C}$ has at most m polygons with a total of $O(n + m^2)$ edges[3]. For each cell $C \in \mathcal{C}$, consider C^*, the set of points that are dual to the lines intersecting C. C^* is a polygonal region, bounded by an infinite convex chain from above and by an infinite concave chain from below. Each vertex of C^* is dual to the line supporting an edge of C. For a pair of polygons $C_1, C_2 \in \mathcal{C}$, an intersection point of the edges of C_1^*, C_2^* is dual to a common tangent of C_1 and C_2. Since C_1, C_2 are disjoint, the boundaries of C_1^*, C_2^* intersect in at most 4 points.

Let us consider the arrangement $\mathcal{A}(\mathcal{C}^*)$ of the polygonal chains bounding the regions C^*, for all $C \in \mathcal{C}$. It has $O(n + m^2)$ complexity, and can be computed in time $O(m^2 + n \log n)$, for instance by Mulmuley's randomized incremental algorithm [18, 6]. This algorithm actually computes the vertical decomposition $\mathcal{A}^{||}(\mathcal{C}^*)$ of the arrangement, together with a point location data structure with $O(\log n)$ expected query time. We use this data structure to locate the points ℓ^* dual to all lines $\ell \in L \setminus R$. From this we can determine, for every ℓ, the regions of $\mathcal{C}^*$ containing ℓ^*, or in other words, the polygons of $\mathcal{C}$ intersecting ℓ. Indeed, after having located all points of the form ℓ^*, we traverse the adjacency graph of the trapezoids in $\mathcal{A}^{||}(\mathcal{C}^*)$. At each trapezoid $\tau \in \mathcal{A}^{||}(\mathcal{C}^*)$ we compute $\mathcal{C}^*(\tau)$, the set of regions that contain the trapezoid $\tau \in \mathcal{A}^{||}(\mathcal{C}^*)$, and output the pairs (ℓ, C) for $\ell^* \in \tau$ and $C^* \in \mathcal{C}^*(\tau)$. Suppose we arrive at τ from τ', then $\mathcal{C}^*(\tau)$ and $\mathcal{C}^*(\tau')$ differ by at most one region (the region whose boundary separates τ from τ'), and thus $\mathcal{C}^*(\tau)$ can be obtained from $\mathcal{C}^*(\tau')$ in $O(1)$ time.

The total time spent in this step is $O(m^2 + n \log n)$ plus the number of polygon/line incidences. The expected number of these incidences is bounded by $O(\kappa(r, m) \cdot (n/r)) = O(n + m^2)$, using Proposition 2.1 with $r = n/t$ and $c = 1$.

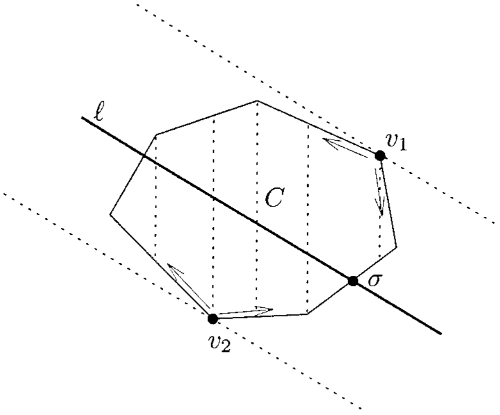


Figure 1: Finding σ.

Step 5. Let C be a cell in $\mathcal{A}(R, P)$, and let $L_C \subseteq L \setminus R$ be the set of lines intersecting the interior of C. For each line $\ell \in L_C$, we compute the trapezoids of $C^{||}$ intersected by ℓ, as follows. Since the lines in L are sorted by their slopes, by being careful in Step 4, we can ensure that the lines of L_C are also sorted by their slopes. For each line $\ell \in L_C$ we compute the two vertices v_1, v_2 of C that support the lines parallel to ℓ (see Figure 1). This can be done, over all lines of L_C, in $O(|L_C|)$ time by merging the slopes of L_C with the slopes of the edges of C; we leave out the easy details for the reader. Next, we traverse ∂C in clockwise as well as counter-clockwise order in a lock-step fashion, starting from both v_1 and v_2 simultaneously (so we preform 4 traversals in a lock-step fashion, as depicted in Figure 1), until we reach an intersection point σ of ℓ and C. Since ℓ intersects C, we will eventually find such an intersection point. Finally, by tracing ℓ through $C^{||}$, starting from σ, we compute all k trapezoids of $C^{||}$

[3] The latter estimate follows from the bound for $\kappa(n, m)$ mentioned in Section 1, in fact it is the weaker bound proved by Canham [4].

that ℓ intersects. The time spent in finding σ and tracing ℓ is easily seen to be $O(k)$. Summing over all cells $C \in \mathcal{A}(R,P)$ and over all lines of L_C, the total time spent is $O(\sum_{\Delta \in \mathcal{A}^{||}(R,P)} w(\Delta))$, whose expected value, by Proposition 2.1 (i), is $O(m^2 + n \log n)$.

Step 6. Let Δ be a trapezoid of $\mathcal{A}^{||}(R,P)$. After having computed L_Δ, we compute the arrangement $\mathcal{A}(L_\Delta)$ using, say, a randomized incremental algorithm. We clip $\mathcal{A}(L_\Delta)$ within Δ, and compute the vertical decomposition of the clipped arrangement. For each point $p \in P \cap \Delta$, we also compute the trapezoid of this vertical decomposition containing p. The time spent in this step is easily seen to be $O(w(\Delta)^2 + |P \cap \Delta| \log w(\Delta))$ per trapezoid $\Delta \in \mathcal{A}^{||}(R,P)$.

For a cell $C \in \mathcal{A}(R,P)$, let $\mathbf{\Delta}_C$ be the set of the resulting trapezoids that lie in C. We now define a graph $\mathcal{G}_C$ on the trapezoids of $\mathbf{\Delta}_C$. The vertices of $\mathcal{G}_C$ are the trapezoids of $\mathbf{\Delta}_C$, and two trapezoids are connected by an edge if they share a vertical edge. By performing a depth first search on $\mathcal{G}_C$, we can extract all connected components of $\mathcal{G}_C$ whose trapezoids contain any point of P. That is, we pick a point $p \in P \cap C$. Let $\tau_p \in \mathbf{\Delta}_C$ be the trapezoid containing p. We perform a depth first search in $\mathcal{G}_C$ starting from τ_p until we find the entire connected component of $\mathcal{G}_C$ containing τ_p. Let $\mathbf{\Delta}_C(p)$ be the set of trapezoids in this component; then the union of these trapezoids is exactly the cell of $\mathcal{A}(L, \{p\})$. The vertices of the cell, sorted in the clockwise order, can be easily obtained by merging the trapezoids of $\mathbf{\Delta}_C(p)$ in an obvious manner.

If there is a point of $P \cap C$ that does not lie in $\mathbf{\Delta}_C(p)$, we repeat the same procedure. We continue this step until we have extracted all component of $\mathcal{G}_C$ that contain any point of $P \cap C$. This gives $\mathcal{A}(L, P \cap C)$.

Repeating this step for all cells of $\mathcal{A}(R,P)$, we obtain all cells of $\mathcal{A}(L,P)$. Finally, we compute the vertical decomposition of all the cells. The total running time for Step 6 is $O(m \log n) + \sum_{\Delta \in \mathcal{A}(R,P)} O(w(\Delta)^2)$, and its expected value is

$$O(m \log n + \kappa(r,m)(n/r)^2) = O(m^2 + n)\,.$$

Putting all the pieces together, the total expected running time of Steps 4–6 is $O(m^2 + n \log n)$. Let $T(n,m)$ denote the maximum expected time of the entire algorithm, then we obtain the following recurrence.

$$T(n,m) \leq \sum_{i=1}^{q} T\left(\lfloor n/t \rfloor, m_i\right) + C(m^2 + n \log n)\,,$$

where $m_i \leq m/\sqrt{t}$ for $i \leq q = \lceil \sqrt{t}\, \rceil$, $\sum_{i=1}^{q} m_i = m$, and C is an appropriate constant. The solution of this recurrence is

$$T(n,m) = O(m^2 + n \log n)\,.$$

If $m > \sqrt{n \log n}$, we can divide the points of P into groups of size $\sqrt{n \log n}$, and solve the subproblems separately. This standard batching technique yields a more convenient bound for the expected running time, namely $O(m\sqrt{n \log n} + n \log n)$. Hence, we can conclude

Lemma 3.1 *Given a set L of n lines and a set P of $m \leq n^2$ points in the plane, the cells of $\mathcal{A}(L)$ containing the points of P can be computed by a randomized algorithm in expected time $O(m\sqrt{n \log n} + n \log n)$.*

We now present another randomized algorithm whose running time is significantly better for larger values of m. Although the basic idea is the same as in [1], the algorithm presented here is simpler because we allow randomization.

We choose a random subset $R \subseteq L$ of size r, where

$$r = \left\lceil \frac{m^{2/3}}{n^{1/3} \log^{1/3}(n/\sqrt{m})} \right\rceil .$$

Using a randomized incremental algorithm, we construct $\mathcal{A}^{||}(R)$ plus a point-location data structure for $\mathcal{A}^{||}(R)$ in expected time $O(r^2)$ [6]. For each trapezoid $\Delta \in \mathcal{A}^{||}(R)$, let $L_\Delta \subseteq L \setminus R$ be the set of lines that intersect the interior of Δ and $P_\Delta \subseteq P$ the set of points that are contained in Δ. L_Δ can be computed in time $O(nr)$ by tracing each line through $\mathcal{A}^{||}(R)$ and P_Δ can be computed in expected time $O(m \log n)$ by locating each point of P in $\mathcal{A}^{||}(R)$. Set $n_\Delta = |L_\Delta|$ and $m_\Delta = |P_\Delta|$. Let Z_Δ be the set of trapezoids in the vertical decomposition of cells of $\mathcal{A}(L_\Delta \cup \{\partial\Delta\})$ that intersect the boundary of Δ and lie inside Δ. Clarkson et al. [9] proved that $|Z_\Delta| = O(n_\Delta)$.

For each trapezoid $\Delta \in \mathcal{A}^{||}(R)$, it is sufficient to compute $\mathcal{A}^{||}(L_\Delta, P_\Delta)$ and Z_Δ. We compute $\mathcal{A}(L_\Delta, P_\Delta)$ in expected time $O(m_\Delta\sqrt{n_\Delta \log n_\Delta} + n_\Delta \log n_\Delta)$ using Lemma 3.1. Z_Δ can be computed by a randomized incremental algorithm. Roughly speaking, we clip the lines of $L_\Delta \setminus R$ within Δ, add the clipped segments one by one in a random order, and maintain the vertical decomposition of the cells of the segments added so far which intersect $\partial\Delta$. Following the same analysis as in [6], it can be shown that the expected running time of the algorithm is $O(n_\Delta \log n_\Delta)$. Hence, the expected running time of the algorithm is

$$\mathrm{E}\left[\sum_{\Delta\in\mathcal{A}^{\|}(R)} O\left(m_\Delta\sqrt{n_\Delta\log n_\Delta}+n_\Delta\log n_\Delta\right)\right]+$$
$$O(nr)+O(m\log n)\,.$$

By a result of Clarkson and Shor [10] (or also by Proposition 2.1), we have

$$\mathrm{E}\Bigl[\sum_{\Delta\in\mathcal{A}^{\|}(R)} n_\Delta\Bigr] \;=\; O(nr) \quad \text{and}$$
$$\mathrm{E}\Bigl[\sum_{\Delta\in\mathcal{A}^{\|}(R)} m_\Delta\sqrt{n_\Delta\log n_\Delta}\Bigr] \;=\; O\left(m\sqrt{\frac{n}{r}\log\frac{n}{r}}\right).$$

Thus, the expected running time of the algorithm is bounded by $O(m\sqrt{\frac{n}{r}\log(n/r)}+nr\log\frac{n}{r}+m\log n)$. Substituting the value r in the above expression, we obtain

Theorem 3.2 *Given a set L of n lines and a set P of m points, the faces of $\mathcal{A}(L)$ containing the points of P can be computed by a randomized algorithm in expected $O(m^{2/3}n^{2/3}\log^{2/3}\frac{n}{\sqrt{m}}+(m+n)\log n)$ time.*

4 Computing cells in segment arrangements

Next, we present an algorithm for computing marked cells in arrangements of segments. Let S be a set of n segments and P a set of m points in the plane. The goal is to compute $\mathcal{A}(S,P)$ and its vertical decomposition $\mathcal{A}^{\|}(S,P)$. Again, we begin by a simpler algorithm which is effective for few cells, and then plug the random sampling technique to handle larger values of m.

The outline of the first algorithm is the same as in the previous section, except that we must now interpret the operations in terms of segments. Since the cells of $\mathcal{A}^{\|}(R,P)$ are not necessarily simply connected, we may have to deal with $m+n$ polygons even though there are only m cells. Consequently, the computation of the sets of cells intersected by each segment of $S\setminus R$ in Step 4 and the computation of S_Δ for each trapezoid $\Delta\in\mathcal{A}^{\|}(R,P)$ in Step 5 now become considerably more complicated. Another difficulty in computing S_Δ is that we now have to detect intersections between simple polygons and segments rather than between convex polygons and lines. In the remainder of this section we will describe how to compute the sets S_Δ.

The boundary of each cell C, ∂C, of $\mathcal{A}(R,P)$ is composed of (at most) one *outer* component and a family of *inner* components such that C lies in the interior of the outer component and in the exterior of each inner component. Each component of ∂C can be regarded as a simple polygonal chain. Let $\mathcal{O}$ be the set of outer boundary components of the cells in $\mathcal{A}(R,P)$, and let $\mathcal{I}$ be the set of the inner boundary components of these cells. We have $|\mathcal{O}|\le m$ and $|\mathcal{I}|\le m+n$. Let μ be the total number of edges of all polygons in $\mathcal{O}\cup\mathcal{I}$; obviously, $\mu\le\eta(n/t,m)$.

We first decompose each segment $g\in S\setminus R$ into maximal subsegments, so that each subsegment lies in the interior of some outer component O, i.e. we cut each segment at the intersection points of $\mathcal{O}$ and S and discard the subsegments that lie in the exterior of $\mathcal{O}$. Let Σ be the set of resulting subsegments. Next, for each subsegment $\sigma\in\Sigma$, we compute the trapezoids of $\mathcal{A}^{\|}(R,P)$ intersected by σ.

Suppose that we have already computed Σ in Step 4. Then in Step 5 we compute S_Δ, for all $\Delta\in\mathcal{A}^{\|}(R,P)$, as follows. We preprocess each polygonal chain $I\in\mathcal{I}$, in linear time, for ray shooting queries, so that the first intersection point of a query ray and I can be computed in logarithmic time, see [5, 15]. The total time spent in preprocessing $\mathcal{I}$ is $O(\mu)=O(\eta(n,m))$.

Let σ be a segment of Σ that lies in the interior of the outer component $O\in\mathcal{O}$ of ∂C. Let a,b be the endpoints of σ, and let $\Delta(a)$ be the trapezoid of $C^{\|}$ containing a. If a is not an endpoint of a segment of $S\setminus R$, then a lies on the boundary of $\Delta(a)$. We check whether $b\in\Delta(a)$. If the answer is 'yes', then $\Delta(a)$ is the only trapezoid of $C^{\|}$ intersected by σ, and we stop. If $b\notin\Delta(a)$, we compute the other intersection point, a_1, of σ and $\Delta(a)$. If a_1 lies on a vertical edge of $\Delta(a)$, we also compute, in constant time, the next trapezoid $\Delta(a_1)$ of $C^{\|}$ intersected by σ. We then repeat the same step with a_1 and $\Delta(a_1)$. if a_1, on the other hand, lies on an edge of the cell C, then a_1 lies on the boundary of some inner component $I\in\mathcal{I}$ of C, and the portion of the segment σ immediately following a_1 lies outside C. Using the ray shooting data structure, we compute the next intersection point a_2 of the polygonal chain I and the segment $\overrightarrow{a_1b}$. Once we know a_2, we can also compute the trapezoid of $C^{\|}$ containing a_2, and we continue tracing σ through $C^{\|}$.

For each trapezoid intersected by σ, we spend $O(\log n)$ time, so the total time spent in computing the k_σ trapezoids intersected by σ is $O(k_\sigma\log n)$. Summing over all segments of Σ, the total time spent is $\sum_{\sigma\in\Sigma}O(k_\sigma\log n) \;=\; O\left(\sum_{\Delta\in\mathcal{A}^{\|}(R,P)}n_\Delta\log n\right)$, where $n_\Delta=|S_\Delta|$.

Next, we describe how to compute the set Σ. Notice that it is sufficient to compute all intersection points between the segments of $S \setminus R$ and the outer polygonal chains in $\mathcal{O}$.

Let J_O be an interval corresponding to the projection of the polygoal chain $O \in \mathcal{O}$ onto the x-axis, and let $\mathcal{J} = \{J_O \mid O \in \mathcal{O}\}$.

We construct an interval tree T on $\mathcal{J}$; see Mehlhorn [17] for details on interval trees. T is a minimum height binary tree with at most $2m$ leaves. Each node v of T is associated with an interval U_v, and a point x_v. Let $W_v = U_v \times [-\infty, +\infty]$ be a vertical strip, and let h_v be the vertical line passing through x_v. For the root u, W_u is the entire plane and h_u is the vertical line passing the middle endpoint of the intervals of $\mathcal{J}$. Each interval $J \in \mathcal{J}$ is stored at the highest node v of T such that $x_v \in J$.

Let $\mathcal{J}_v$ be the set of intervals stored at v. We associate two subsets $\mathcal{O}_v, Z_v$ of $\mathcal{O}$ with v. Let $\mathcal{O}_v = \{O \mid J_O \in \mathcal{J}_v\}$ and $Z_v = \bigcup_w \mathcal{O}_w$, where the union is taken over all descendants of v, including v; set $m_v = |\mathcal{O}_v|$ and $z_v = |Z_v|$. Finally, let μ_v (resp. ζ_v) denote the total number of edges in $\mathcal{O}_v$ (resp. Z_v). Since each polygonal chain of $\mathcal{O}$ appears in exactly one $\mathcal{O}_v$, we have $\sum_{v \in T} \mu_v = \mu$ and $\sum_{v \in T} \zeta_v \leq 2\mu \log m$. Moreover, it can be shown that if v_1, v_2 are the children of v then $z_{v_1}, z_{v_2} \leq z_v/2$, which implies that $\sum_{v \in T} z_v^2 = O(m^2)$.

Since the polygonal chains in $\mathcal{O}_v$ are pairwise disjoint and all of them intersect a vertical line, we can regard $\mathcal{O}_v$ along with appropriate portions of the vertical line h_v as a simple polygon Π_v, and preprocess Π_v in $O(\mu_v)$ time for answering ray shooting queries. Using this data structure, one can report all k intersection points of a segment g and $\mathcal{O}_v$ in time $O((k+1)\log \mu_v)$.

Next, we take the convex hull of each polygonal chain in Z_v, and preprocess the resulting convex polygons into a data structure, as described in the previous section, so that all convex polygons intersected by a query line can be reported quickly. Since any two polygonal chains of $\mathcal{O}$ are disjoint, the boundaries of their convex hulls intersect in at most two points, and so they have at most 4 common tangents. Consequently, the line intersection searching structure has size $O(z_v^2 + \zeta_v)$. Moreover, it can be computed in time $O(z_v^2 + z_v \log \zeta_v + \zeta_v)$, using the algorithm of [19]. We also preprocess each $O \in \mathcal{O}$ in linear time for ray shooting queries as in [15]. It can be shown that the total preprocessing time is $O(m^2 + \sum_v (z_v \log \zeta_v + \zeta_v)) = O(m^2 + m \log m \log n + \mu \log m)$. We omit the details.

Let $g \in S \setminus R$ be a segment. All intersection points of g and $\mathcal{O}$ can be computed as follows. We search the tree T with g starting from the root. Let v be a node visited by the query procedure. If the endpoints of g do not lie in the vertical strip W_v, i.e., g completely crosses W_v, then g intersects $O \in Z_v$ if and only if the line supporting g intersects the convex hull of O. Thus, we first compute all polygonal chains of Z_v intersected by g, using the line intersection searching structure, and then, for each $O \in Z_v$ intersected by g, we compute the intersection points of g and O using the ray shooting data structure. If k_g^v is the number of intersection points between g and the polygonal chains of Z_v, then the total time in reporting these intersections is $O((k_g^v + 1) \log \zeta_v)$.

If one of the endpoints of g lies in W_v, we can compute all a_g^v intersection points between $\mathcal{O}_v$ and g in time $O((a_g^v + 1) \log \mu_v)$, using the ray shooting data structure for $\mathcal{O}_v$. Let v_1, v_2 be the children of the node v. If g intersects W_{v_1} (resp. W_{v_2}), we visit v_1 (resp. v_2). It is easily shown that the query procedure visits $O(\log m)$ nodes, and the query time is $O((\log m + k_g) \log n)$, where k_g is the total number of intersection points reported.

We repeat this procedure for all segments $g \in S \backslash R$. Since $\mu \leq \eta(n, m) = O(m^2 + n(\log m + \alpha(n)))$ and $\sum_{g \in S \backslash R} k_g \leq \sum_{\Delta \in \mathcal{A}^{||}(R,P)} n_\Delta$, the total cost of computing the intersection points is $O((m^2 + n \log m + n\,\alpha(n)) \log n + \sum_{\Delta \in \mathcal{A}^{||}(R,P)} n_\Delta \log n)$. As in the previous section, the time spent in Step 6 (refining the cells of $\mathcal{A}^{||}(R, P)$) is $O(\sum_\Delta n_\Delta^2)$. Using Proposition 2.1 (ii), we obtain that the total expected time spent in the merge step is $O\left((m^2 + n \log m + n\,\alpha(n)\right) \log n)$.

Following the same analysis as earlier, we can conclude that the total running time of the first algorithm for computing $\mathcal{A}(S, P)$ is $O\left((m^2 + n \log m + n\,\alpha(n)) \log n\right)$. We can again use the batching technique if m is large. Omitting the details, we obtain

Lemma 4.1 *Given a set S of n segments and a set P of m points, the faces of $\mathcal{A}(S)$ that contain a point of P can be computed by a randomized algorithm in expected time $O((m\sqrt{n \log m} + n(\log m + \alpha(n))) \log n)$.*

For larger values of m, we again use the random sampling technique as in the previous section. That is, we choose a random subset $R \subseteq S$ of size $r = \left\lceil \frac{m^{2/3}}{n^{1/3}} \cdot \frac{\log^{1/3}(n/\sqrt{m})}{\alpha^{2/3}(n/\sqrt{m})} \right\rceil$, and compute $\mathcal{A}^{||}(R)$. For each $\Delta \in \mathcal{A}^{||}(R)$, we compute $P_\Delta = P \cap \Delta$ and S_Δ, the set of segments that intersect Δ. We clip the segments within Δ. The total time spent in this step is $O(r^2 + (m + nr) \log r)$. Let z be a point lying in the unbounded face of $\mathcal{A}(S)$. For each

$\Delta \in \mathcal{A}^{\|}(R)$, we compute $\mathcal{A}^{\|}(S_\Delta, P_\Delta \cup \{z\})$, in time $O((m_\Delta\sqrt{n_\Delta \log m_\Delta} + n_\Delta(\log m_\Delta + \alpha(n_\Delta)))\log n_\Delta)$, using Lemma 4.1, and then glue them together. We omit the rather routine details from here. The overall expected running time of the algorithm is

$$\mathrm{E}\Big[\sum_{\Delta\in\mathcal{A}^{\|}(R)} O((n_\Delta(\alpha(n_\Delta)+\log m_\Delta)+ m_\Delta\sqrt{n_\Delta \log m_\Delta}\Big] + O((m+nr)\log r)\,.$$

Again, using the results by Clarkson-Shor [10] and substituting the value of r, we obtain

Theorem 4.2 *Given a set S of n segments and a set P of m points, the faces of $\mathcal{A}(S)$ that contain a point of P can be computed by a randomized algorithm in expected time $O(m^{2/3}n^{2/3}\log^{4/3}\frac{n}{\sqrt{m}}\,\alpha^{1/3}(\frac{n}{\sqrt{m}})+(m+n\log m + n\,\alpha(n))\log n)$.*

Finally, let us remark that if $\mathcal{A}(S)$ is sparse, that is, if it has only $k = o(n^2)$ vertices, then using the fact that the expected number of trapezoids in $\mathcal{A}^{\|}(R)$ is $O(kr^2/n^2 + r)$, we can do a more careful analysis, choose $r = \left\lceil n\left(\frac{m}{k}\right)^{2/3}\frac{\log^{1/3}(k/m)}{\alpha^{2/3}(k/m)}\right\rceil$, and can show that the expected running time of the algorithm is $O(m^{2/3}k^{1/3}\log^{4/3}\frac{k}{m}\alpha^{1/3}(\frac{k}{m})+(m+n\log m+ n\,\alpha(n))\log n)\,.$

Acknowledgments. The authors thank Mark de Berg, Mark Overmars, and Micha Sharir for several useful discussions.

References

[1] P. Agarwal, Partitioning arrangements of lines: II. Applications, *Discrete and Computational Geometry* 5 (1990), 533–573.

[2] B. Aronov, H. Edelsbrunner, L. Guibas and M. Sharir, Improved bounds on the complexity of many faces in arrangements of segments, *Combinatorica*, 12 (1992), 261–274.

[3] M. de Berg, K. Dobrindt and O. Schwarzkopf, On lazy randomized incremental construction, to appear in *Proceedings 26th Annual ACM Symposium on Theory of Computing*, 1994.

[4] R. Canham, A theorem on arrangements of lines in the plane, *Israel J. Math.* 7 (1969), 393–397.

[5] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir and J. Snoeyink, Ray shooting in polygons using geodesic triangulations, *Proc. 17th Int. Colloq. Automata, Languages and Programming*, 1991, pp. 661–673.

[6] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir and J. Snoeyink, Computing a face in an arrangement of line segments, *SIAM J. Computing* 22 (1993), 1286–1302.

[7] B. Chazelle and J. Friedman, A deterministic view of random sampling and its use in geometry, *Combinatorica* 10 (1990), 229–249.

[8] K. Clarkson, Computing a single face in an arrangement of segments, 1990, manuscript.

[9] K. Clarkson, H. Edelsbrunner, L. Guibas, M. Sharir and E. Welzl, Combinatorial complexity bounds for arrangements of curves and spheres, *Discrete and Computational Geometry* 5 (1990), 99–160.

[10] K. Clarkson and P. Shor, Applications of random sampling in computational geometry II, *Discrete and Computational Geometry* 4 (1989), 387–421.

[11] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.

[12] H. Edelsbrunner, L. Guibas and M. Sharir, The complexity of many faces in arrangements of lines and of segments, *Discrete and Computational Geometry* 5 (1990), 161–196.

[13] H. Edelsbrunner and E. Welzl, On the maximal number of edges of many faces in an arrangement, *Journal of Combinatorial Theory, Series A* 41 (1986), 159–166.

[14] L. Guibas and M. Sharir, Combinatorial and algorithms of arrangements, in *New Trends in Discrete and Computational Geometry* (J. Pach, ed.), Springer-Verlag, New York-Berlin-Heidelberg, 1993, 9–36.

[15] J. Hershberger and S. Suri, A pedestrian approach to ray shooting: Shoot a ray, take a walk, *Proc. 4th ACM-SIAM Symp. Discrete Algorithms*, 1993, pp. 54–63.

[16] D. Haussler and E. Welzl, Epsilon-nets and simplex range queries, *Discrete Comput. Geom.* 2 (1987), 127–151.

[17] K. Mehlhorn, *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, Springer-Verlag, Berlin, 1984.

[18] K. Mulmuley, A fast planar partition algorithm, I, *J. Symbolic Computation* 10 (1990), 253–280.

[19] K. Mulmuley, A fast planar partition algorithm, II, *J. ACM* 38 (1991), 74–103.

[20] E. Szemerédi and W. Trotter Jr., Extremal problems in discrete geometry, *Combinatorica* 3 (1983), 381–392.

Matching Shapes with a Reference Point

Helmut Alt*
Freie Universität Berlin
Fachbereich Mathematik und Informatik
Takustraße 9, D-14195 Berlin, Germany
e-mail: alt@inf.fu-berlin.de

Oswin Aichholzer†
Institut für Grundlagen der Informationsverarbeitung
Technische Universität Graz
Schießstattgasse 4, A-8010 Graz, Austria
e-mail: oaich@igi.tu-graz.ac.at

Günter Rote
Institut für Mathematik, Technische Universität Graz
Steyrergasse 30, A-8010 Graz, Austria
e-mail: rote@ftug.dnet.tu-graz.ac.at

Abstract

For two given point sets, we present a very simple (almost trivial) algorithm to translate one set so that the Hausdorff distance between the two sets is not larger than a constant factor times the minimum Hausdorff distance which can be achieved in this way. The algorithm just matches the so-called Steiner points of the two sets.

The focus of our paper is the general study of reference points (like the Steiner point) and their properties with respect to shape matching.

For more general transformations than translations, our method eliminates several degrees of freedom from the problem and thus yields good matchings with improved time bounds.

1 Introduction

This work is motivated by a problem that is typical in application areas such as computer vision or pattern recognition, namely, given two figures A, B, to determine how much they "resemble each other".

Here, a "figure" will be a union of finitely many points and line segments in $\mathbb{R}^2$ or triangles in $\mathbb{R}^3$. Note that sets of curves in $\mathbb{R}^2$ and $\mathbb{R}^3$ or surfaces in $\mathbb{R}^3$ can be approximated arbitrarily closely by these objects. As a measure for "resemblance" we will use the *Hausdorff-metric* δ_H, which is a somehow natural distance measure and gives reasonable results in practice (see [HKR]). It is defined as follows:

Definition 1 *Let A, B be compact subsets of $\mathbb{R}^2$ or $\mathbb{R}^3$. Then*

$$\widetilde{\delta_H}(A,B) := \max_{a\in A} \min_{b\in B} \|a-b\|,$$

where $\|\cdot\|$ is the Euclidean norm and the Hausdorff-distance is defined as

$$\delta_H(A,B) := \max\{\widetilde{\delta_H}(A,B), \widetilde{\delta_H}(B,A)\}.$$

If A and B consist of n and m line segments in the plane, their Hausdorff-distance $\delta_H(A,B)$ can be computed in time $O((n+m)\log(n+m))$ (cf. [ABB]). However, it is more natural to assume that A and B are not fixed but can be moved by a *translation*, by a *rigid motion* (translation and rotation) or even transformed by a *similarity* (scaling and rigid motion) in order to match them as well as possible and then determine the minimal Hausdorff-distance. So, in general, we have a set $\mathcal{T}$ of allowed transformations and want to determine for given figures A and B:

$$\min_{T\in\mathcal{T}} \delta_H(A, T(B)).$$

*This research was supported by the ESPRIT Basic Research Action Program No. 7141, Project ALCOM II. It was initiated during a stay of the author at the TU Graz sponsored by funds for the promotion of foreign relations of that university.

†That research was supported by the Jubiläumsfond der Oesterreichischen Nationalbank.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

Note that, for similarities, it makes a difference if we exchange the sets A and B in this problem.

This problem of finding an optimal matching has been considered for the two-dimensional case in several previous articles: In [ABB], an algorithm of running time $O((nm)\log(nm)\log^*(nm))$ for the case that $\mathcal{T}$ is the set of translations along one fixed direction is described. An $O((nm)^2\log^3(nm))$ algorithm for arbitrary translations is presented in the paper [AST], which can be improved to $O(nm(n+m)\log(nm))$ if A and B are finite sets of points [HKS]. The paper [CGHKKK] gives an algorithm of time complexity $O((nm)^3\log^2(nm))$ for arbitrary rigid motions. The latter two algorithms use sophisticated and powerful tools like parametric search and therefore do not seem to be applicable in practice.

Here, we follow a different approach which was already used in [ABB]. We do not try to find an *optimal* solution but an *approximation* to the optimal one by simpler algorithms. More precisely, if the optimal matching yields Hausdorff-distance δ our algorithms will find a matching T such that

$$\delta_H(A, T(B)) \leq a\delta$$

for some constant $a > 1$. We call such a solution a *pseudooptimal matching* with *loss factor a*.

A somehow similar approach has been taken in several articles by Heffernan and Schirra ([H,HS]) considering matching of finite sets A, B of points under rigid motions. They consider both, the *decision problem*, whether for a given ε A (or a subset of a given size k) can be mapped in a 1-1-fashion into the ε-neighborhoods of points of B, and the *optimization problem*, namely to find the smallest ε of this kind. Both problems are solved very efficiently by allowing that the optimal ε is not found exactly, but within a small tolerance interval (*approximate* algorithms).

The aim of this paper is to work out the general idea of using "reference points" for pseudooptimal algorithms. Reference points were first introduced in [ABB] and used in [CKS] to obtain an O(mn log(mn)) approximate algorithm for the decision problem for sets of line segments. Here we will also present a reference point that gives better bounds than the one in [ABB] and can be applied to similarities and problems in three dimensions as well.

2 Reference Point Methods

Like in [ABB] pseudooptimal algorithms use suitable *reference points*, which we define for arbitrary dimension d as follows:

Definition 2 *Let C^d denote the set of compact subsets of $\mathbb{R}^d$, and let $\mathcal{T}$ be a set of transformations on $\mathbb{R}^d$. A mapping $s\colon C^d \to \mathbb{R}^d$ is called a* reference point *with respect to $\mathcal{T}$ if the following two conditions are satisfied:*

(a) s is equivariant *with respect to $\mathcal{T}$, i.e., for all $A, B \in C^d$and $T \in \mathcal{T}$ we have*

$$s(T(A)) = T(s(A)).$$

(b) There exists some constant $c \geq 0$ such that for all $A, B \in C^d$,

$$\|s(A) - s(B)\| \leq c \cdot \delta_H(A, B).$$

In other words, s is a Lipschitz-continuous mapping between the metric spaces (C^d, δ_H) and $(\mathbb{R}^d, \|\cdot\|)$ with Lipschitz constant c. We call c the quality *of the reference point s.*

Based on the existence of a reference point for $\mathcal{T}$ we obtain the following algorithms for pseudooptimal matchings where $\mathcal{T}$ is the set of translations, rigid motions, and similarity transformations, respectively:

Algorithm T

1. Compute $s(A)$ and $s(B)$ and translate B by $s(A) - s(B)$ (so that $s(B)$ is mapped onto $s(A)$). Let B' be the image of B.
2. Output B' as the pseudooptimal solution (together with the Hausdorff-distance $\delta_H(A, B')$).

Algorithm R

1. as in Algorithm T.
2. Find an optimal matching of A and B' under rotations of B' around $s(A)$.
3. Output the solution B'' and the Hausdorff-distance $\delta_H(A, B'')$.

Algorithm S

1. as in Algorithm T.
2. determine the diameters $d(A)$ and $d(B)$ and scale B' by $\alpha := d(A)/d(B)$ around the center $s(A)$.
3. as Step 2 in algorithm R with the scaled image of B'.
4. as Step 3 in algorithm R.

As the algorithms R and S are formulated, they look only for *proper* rigid motions and positive similarities, respectively. Reflections can be included by simply running the algorithm a second time with a reflected copy of A.

These algorithms are simpler than the ones for finding the optimal solutions, since after Step 1 the matchings are restricted to ones leaving the reference point invariant. In d dimensions this eliminates d degrees of freedom. The qualities and running times of these algorithms are as follows:

Theorem 1 *Suppose that we can determine a reference point of quality c for the sets of transformations $\mathcal{T}$ in the Algorithms T, R, and S in linear time. In the case of similarity transformations also assume that $s(A)$ always lies within the convex hull conv(A).*

(a) Algorithm T finds a pseudooptimal matching for translations with loss factor $a = c + 1$.

(b) Algorithm R finds a pseudooptimal matching for rigid motions with loss factor $a = c + 1$.

(c) Algorithm S finds a pseudooptimal matching for similarity transformations with loss factor $a = c + 3$.

In the plane, the running times for two sets of n and m points and line segments are $O((n + m)\log(n+m))$ for Algorithm T and $O(nm\log(nm)\log^*(nm))$ for Algorithms R and S. In three dimensions, where A and B are sets of triangles, the running times become $O(mn)$ for Algorithm T and $O((nm)^3 H(n,m))$ for Algorithms R and S. Here $H(n,m)$ is the time to compute the Hausdorff distance.

Note that an upper bound of $O((n^2m + nm^2) log^2(nm))$ for $H(n,m)$ is known [AG].

For the proof of the theorem we need the following lemmas, which can be shown by elementary geometrical considerations.

Lemma 2 *Let $B \subset \mathbb{R}^d$ be a compact set with diameter $d(B)$, and let p be a point in its convex hull conv(B). Let τ_1, τ_2 be homotheties (scalings) with center p and ratios (scaling factors) α_1 and α_2, respectively. Then*

$$\delta_H(\tau_1(B), \tau_2(B)) \leq |(\alpha_1 - \alpha_2)d(B)|.$$

Lemma 3 *If $A, B \subset \mathbb{R}^d$ are compact sets with diameters $d(A)$ and $d(B)$, respectively, then*

$$|d(A) - d(B)| \leq 2\delta_H(A, B).$$

Proof: This holds because B is contained in the δ-neighborhood of A and vice versa. □

Proof of Theorem 1: We prove only (c) which implicitly contains the proofs for (a) and (b). Consider an optimal similarity transformation S_{opt}. It can be written as $S_{\text{opt}} = \tau_{\text{opt}} \circ T_{\text{opt}}$, where T_{opt} a rigid motion and τ_{opt} is a homothety with ratio α'. Let δ be the optimal Hausdorff-distance $\delta = \delta_H(A, S_{\text{opt}}(B))$. Then

$$\|s(A) - s(S_{\text{opt}}(B))\| \leq c\delta.$$

Let t be the translation by $s(A) - s(S_{\text{opt}}(B))$; then $\widetilde{S} = t \circ S_{\text{opt}}$ is a similarity transformation mapping $s(B)$ onto $s(A)$ and

$$(1) \qquad \delta_H(A, \widetilde{S}(B)) \leq (c+1)\delta.$$

Write $\widetilde{S}$ as $\widetilde{S} = \widetilde{\tau} \circ \widetilde{T}$, where $\widetilde{T}$ a rigid motion mapping $s(B)$ onto $s(A)$ and $\widetilde{\tau}$ is a homothety with center $s(A)$ and ratio α'. Let $\alpha = d(A)/d(B)$ as in Algorithm S, τ the homothety with center $s(A)$ and ratio α, and $S = \tau \circ \widetilde{T}$. Then

$$(2)\ \delta_H(A, S(B)) \leq \delta_H(A, \widetilde{S}(B)) + \delta_H(\widetilde{S}(B), S(B))$$

Now,

$$\begin{aligned}
(3)\ \delta_H(\widetilde{S}(B), S(B)) &= \delta_H\left(\widetilde{\tau}(\widetilde{T}(B)),\ \tau(\widetilde{T}(B))\right) \\
&\leq \left|(\alpha' - \alpha)d(\widetilde{T}(B))\right|, \text{ by Lemma 2} \\
&= |(\alpha' - \alpha)d(B)| = |\alpha' d(B) - d(A)| \\
&= |d(S_{\text{opt}}(B)) - d(A)|, \\
&\qquad \text{(since } \alpha' \text{ was the ratio of } S_{\text{opt}}) \\
&\leq 2 \cdot \delta_H(S_{\text{opt}}(B), A) = 2\delta,
\end{aligned}$$

by Lemma 3. From (1), (2), and (3) we have

$$(4) \qquad \delta_H(A, S(B)) \leq (c+3)\delta$$

for a similarity transformation S composed of a rigid motion that maps $s(B)$ onto $s(A)$ and a homothety with center $s(A)$ and ratio α. Since Algorithm S finds the optimum among these similarity transformations the bound (4) holds for it, as well.

For the time bound we observe that Step 1 can be done in linear time, and Step 2 in $O(n \log n + m \log m)$. (In three dimensions, the diameters can

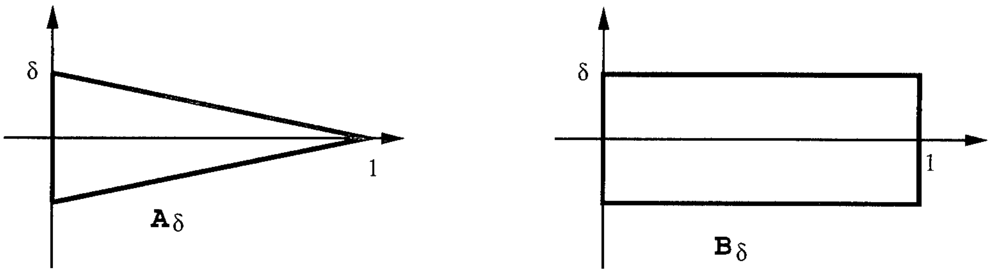


Figure 1: $s(A_\delta) = (1/3, 0)$, $s(B_\delta) = (1/2, 0)$, where s is either the center of the smallest enclosing ellipse or the centroid of the figure. This can be seen by applying an affine transformation which maps A_δ to an equilateral triangle or B_δ to a square.

be computed trivially in $O(n^2 + m^2)$ time.) Step 3, finding the optimal matching under rotations, can be done in time $O(nm \log(nm) \log^*(nm))$ in the plane by the algorithm of [ABB], whose analysis relies on Davenport-Schinzel sequences. This algorithm explicitly computes the Hausdorff-distance, so Step 4 is for free. In 3-space, we have rotations in $\mathbb{R}^3$ around a fixed center, which is an optimal matching problem with 2 degrees of freedom. It can be solved in time $O((nm)^3 H(n,m))$ by methods of [ABB]. □

3 The Steiner Point

The previous section would be useless if it were not possible to find suitable reference points. In [ABB] it was observed that in the two-dimensional case the point $s(A) = (x_{\max}, y_{\max})$, where $x_{\max}$ and $y_{\max}$ are the maximal x- and y-coordinates of points in A, is a reference point of quality $\sqrt{2}$ for translations.

For rigid motions the situation is not as easy. We will first list a few points that come to mind but turn out not to be reference points. In fact, for arbitrary small $\delta > 0$, we can construct figures A_δ, B_δ for which $\delta_H(A_\delta, B_\delta) \leq \delta$, but $\|s(A_\delta) - s(B_\delta)\|$ is not in $O(\delta)$ or does not even converge to 0 for $\delta \to 0$. Some of these counterexamples are from [B], some are new:

$s(A)$ is the centroid of the	$\|s(A_\delta) - s(B_\delta)\|$
a) vertices of the convex hull	$\Omega(1)$
b) convex hull	$\Omega(1)$
c) smallest enclosing circle	$\Omega(\sqrt{\delta})$
d) smallest enclosing ellipse	$\Omega(1)$
e) smallest enclosing rectangle	$\Omega(1)$

Figure 1, for example, shows A_δ, B_δ for cases b) and d). In [ABB] also a positive example of a reference point for rigid motions in two dimensions was given: the centroid of the boundary of the convex hull. It was shown that this reference point is of quality at most $4\pi + 3 \approx 15.57$. Here, we will give a reference point which works even for similarity transformations, is easy to compute, can be generalized to higher dimensions, and whose quality is $4/\pi \approx 1.27$.

First we observe that we can without loss of generality restrict our attention to *convex* figures. In fact, in [ABB] it was shown that for any two compact sets A, B:

$$\delta_H(conv(A), conv(B)) \leq \delta_H(A, B).$$

From that it follows easily that a reference point for the convex hull of a compact set A is a reference point for A, as well.

Our candidate for a reference point is the so-called *Steiner point*, which has been investigated intensively in the field of convex geometry [G, S, Sch].

Definition 3 *We denote by B^d the d-dimensional unit ball and by S^{d-1} its boundary, the $(d-1)$-dimensional unit sphere in $\mathbb{R}^d$.*

Let A be a convex body (convex and compact subset) in $\mathbb{R}^d$. The support function *$h_A: \mathbb{R}^d \to \mathbb{R}$ of A is given by*

$$h_A(\mathbf{u}) = \max_{\mathbf{a} \in A} \langle \mathbf{a}, \mathbf{u} \rangle$$

(see Figure 2).

The Steiner point *$s(A)$ of A is defined as*

$$s(A) = \frac{d}{\mathrm{Vol}(S^{d-1})} \int_{S^{d-1}} h_A(\mathbf{u})\mathbf{u} \, d\omega(\mathbf{u}),$$

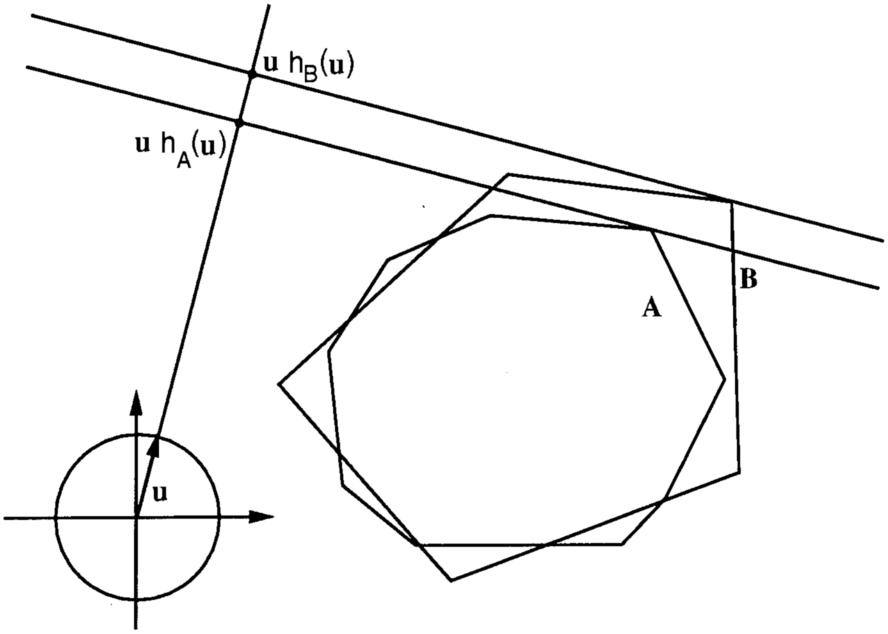


Figure 2: The support functions $h_A(\mathbf{u})$, $h_B(\mathbf{u})$ of two convex bodies A, B.

where $d\omega(\mathbf{u})$ is the surface element of S^{d-1}.

For a non-convex compact set $A \in \mathbb{R}^d$, we define the support function and hence the Steiner point by the same definitions. They coincide with the support function and the Steiner point of the convex hull conv(A).

For the Steiner point we obtain

Theorem 4 *The Steiner point is a reference point for similarity transformations in arbitrary dimension d. For $d = 2$, its quality is $4/\pi$, for $d = 3$ it is $3/2$, for arbitrary d it is less than $\sqrt{2/\pi} \cdot \sqrt{d+1}$.*

Proof: The equivariance of the Steiner point under similarity transformations is well known [G, Sch]. For the bound on the quality, we observe that for two convex bodies A and B, $\|h_A(\mathbf{u}) - h_B(\mathbf{u})\| \leq \delta_H(A,B) =: \delta$ for any $\mathbf{u} \in S^{d-1}$ (see Figure 2).

Now let $\mathbf{p} = s(A) - s(B)$, and consider the inner product of $\mathbf{p}$ with an arbitrary unit vector $\mathbf{e}$. Without loss of generality we assume that $\mathbf{e} = (0, \ldots, 0, 1)$ is the unit vector in the d-th coordinate direction.

$$\langle \mathbf{p}, \mathbf{e} \rangle = \langle s(A) - s(B), \mathbf{e} \rangle$$

$$= \frac{d}{\mathrm{Vol}(S^{d-1})} \int_{S^{d-1}} (h_A(\mathbf{u}) - h_B(\mathbf{u})) \langle \mathbf{u}, \mathbf{e} \rangle \, d\omega(\mathbf{u})$$

$$\leq \frac{d}{\mathrm{Vol}(S^{d-1})} \cdot \Bigg[\int_{\substack{S^{d-1}\\ u_d \geq 0}} (+\delta) \langle \mathbf{u}, \mathbf{e} \rangle \, d\omega(\mathbf{u}) + \int_{\substack{S^{d-1}\\ u_d \leq 0}} (-\delta) \langle \mathbf{u}, \mathbf{e} \rangle \, d\omega(\mathbf{u}) \Bigg]$$

$$= \delta d \cdot \Bigg[\frac{1}{\mathrm{Vol}(S^{d-1})/2} \cdot \int_{\substack{S^{d-1}\\ u_d \geq 0}} \langle \mathbf{u}, \mathbf{e} \rangle \, d\omega(\mathbf{u}) \Bigg]$$

The expression in brackets in the last line is nothing but the d-th coordinate of the center of gravity of the upper unit half-sphere. If we compute the integral by projecting away the d-th coordinate and integrate over $(x_1, x_2, \ldots, x_{d-1}) \in B^{d-1} \subseteq \mathbb{R}^{d-1}$, a straightforward calculation gives that the surface element is transformed by

$$d\omega(\mathbf{u}(x_1, x_2, \ldots, x_{d-1})) = \frac{1}{\langle \mathbf{u}, \mathbf{e} \rangle} dx_1 dx_2 \cdots dx_{d-1}.$$

Thus the integral turns out to be just the volume of B^{d-1}, and we get the following bound on $\langle \mathbf{p}, \mathbf{e} \rangle$:

$$\langle \mathbf{p}, \mathbf{e} \rangle \leq \delta d \cdot \frac{\mathrm{Vol}(B^{d-1})}{\mathrm{Vol}(S^{d-1})/2} = \delta 2d \cdot \frac{\pi^{(d-1)/2} / \Gamma\left(\frac{d+1}{2}\right)}{d\pi^{d/2} / \Gamma\left(\frac{d+2}{2}\right)}$$

$$= \delta \cdot \frac{2}{\sqrt{\pi}} \cdot \frac{\Gamma\left(\frac{d+2}{2}\right)}{\Gamma\left(\frac{d+1}{2}\right)},$$

using the formulas for the volume and surface area of higher-dimensional balls and spheres. This last expression is a bound on the length of $\mathbf{p}$ since $\mathbf{e}$ was in fact an arbitrary unit vector and therefore can be substituted by $\mathbf{p}/\|\mathbf{p}\|$. The values $\Gamma(3/2) = \sqrt{\pi}/2$, $\Gamma(2) = 1$, and $\Gamma(5/2) = 3\sqrt{\pi}/4$ give the claimed quality bounds for $d = 2$ and $d = 3$. The quotient of the two Γ-functions is between $\sqrt{d/2}$ and $\sqrt{(d+1)/2}$, and this gives the general bound. □

From this proof one can see how to construct an example showing that the bound cannot be improved: S^{n-1} must be divided into two half-spheres, and $h_A(\mathbf{u}) - h_B(\mathbf{u})$ will ideally always be equal to $+\delta$ or $-\delta$, depending on the half-sphere in which $\mathbf{u}$ lies. Figure 3 shows two two-dimensional point sets A_δ and B_δ. A_δ is just a circle of radius r, and B_δ consists of a "distorted" circle and an additional point. If we allow to apply any similarity to B_δ in order to minimize the Hausdorff distance from A_δ, the optimal position is as shown in Figure 3, and the Hausdorff distance is δ. The distance of the Steiner point $s(B_\delta)$ from the center $s(A_\delta)$ of the circle can be calculated as $2/\pi \cdot \sqrt{r\delta}\arccos\frac{r-\delta}{r+\delta}$, which approaches $4/\pi \cdot \delta$ as r goes to ∞. If one lets the two Steiner points coincide, the Hausdorff distance rises by this amount, showing that $1+4/\pi$ is indeed the loss factor of Algorithm T. Since A_δ is rotation-symmetric, this holds also for Algorithm R. The above construction generalizes easily to higher dimensions.

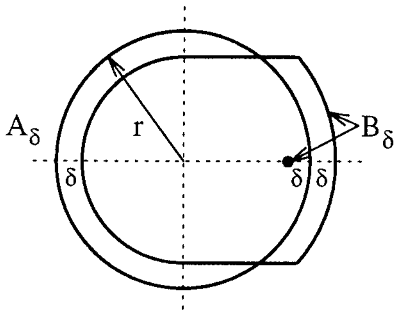


Figure 3: $\|s(A_\delta) - s(B_\delta)\|$ is close to $4/\pi \cdot \delta$.

The following theorem is well-known [G,S].

Theorem 5 *The Steiner point of a convex polytope is the weighted sum of its vertices, where the weight of vertex v is that fraction of the surface of the unit sphere that lies between the unit vectors normal to the hyperplanes meeting at v (the normalized exterior angle at v). (See Figure 4 for a two-dimensional example.)*

For smooth convex bodies, the Steiner point can also be defined as the centroid of a non-uniform mass distribution on the boundary, where the density is the (Gaussian) curvature.

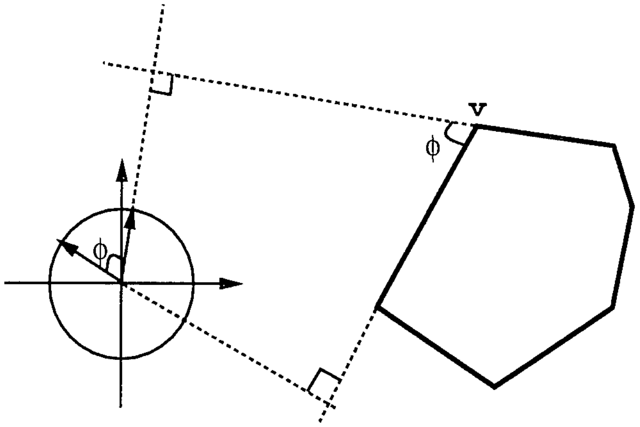


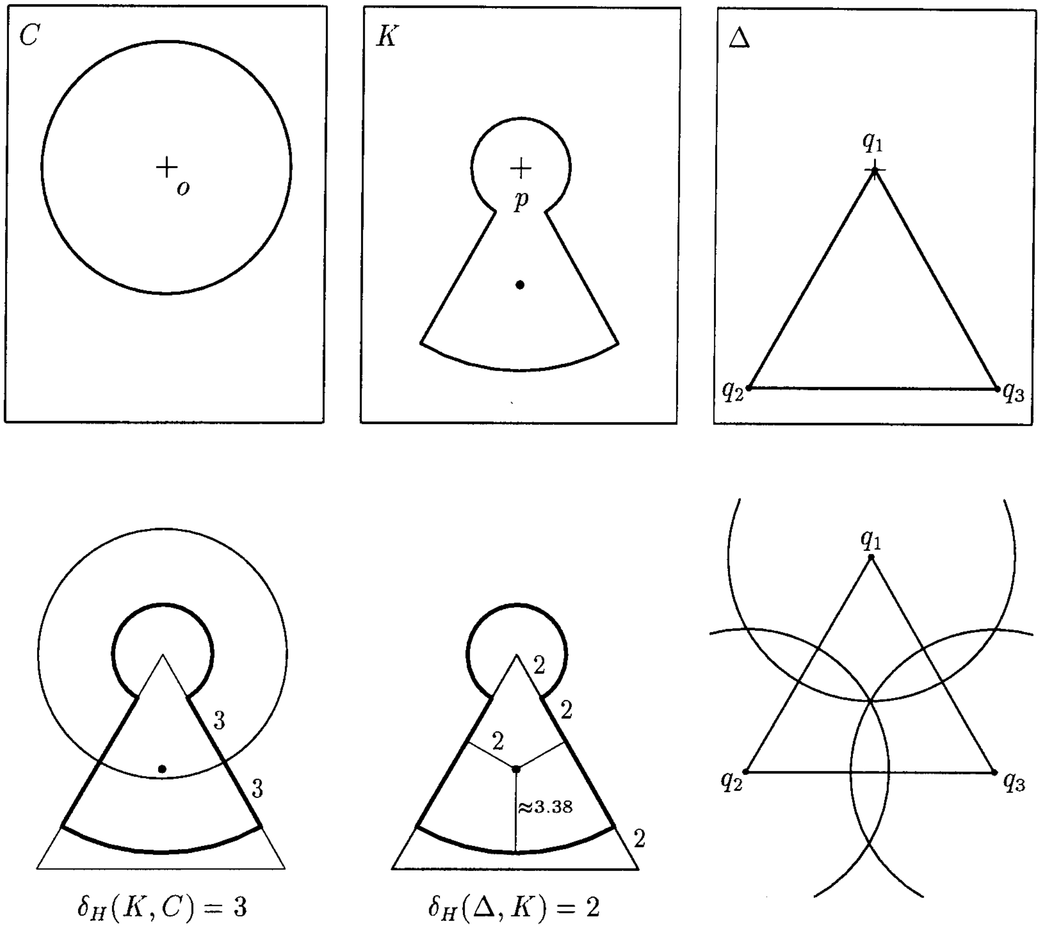
Figure 4: The weight of vertex v is $\frac{\phi}{2\pi}$.

Combining Theorems 1, 4, and 5 we get:

Theorem 6 *Let A and B be sets of n and m line segments in $d = 2$ dimensions or n and m triangles in $d = 3$ dimensions. Then pseudooptimal matchings can be found for A and B applying the corresponding algorithms of Section 2 as indicated in the following table.*

$\mathcal{T}$	running time	loss factor
translations		
$d = 2$	$O((n+m)\log(n+m))$	$4/\pi + 1$
$d = 3$	$O(H(n,m))$	2.5
rigid motions		
$d = 2$	$O(nm\log(nm)\log^*(nm))$	$4/\pi + 1$
$d = 3$	$O((nm)^3 H(n,m))$	2.5
similarities		
$d = 2$	$O(nm\log(nm)\log^*(nm))$	$4/\pi + 3$
$d = 3$	$O((nm)^3 H(n,m))$	4.5

Proof: For the proof note that the Steiner point for a convex polygon or polytope can be computed in linear time because of Theorem 5, after the convex hulls have been constructed in $O(n\log n + m\log m)$ time. The bound of $O(H(n,m))$ (as in



$\delta_H(K, C) = 3$ $\delta_H(\Delta, K) = 2$

Figure 5: A lower bound for the quality of a reference point.

Theorem 1) for translations in three dimensions comes from the final computation of the Hausdorff distance. Just finding the pseudooptimal translation takes only $O(n \log n + m \log m)$ time. □

4 Lower Bounds

For translations in two dimensions the following lower bound on the quality of reference points can be shown:

Theorem 7 *The quality c of any reference point for translations (Algorithm T) in two (or higher) dimensions cannot be better (i.e., smaller) than* $\sqrt{4/3}$.

Proof: We consider three point sets (see the upper part of Figure 5): A circle C with center o and radius 5; a keyhole-shaped figure K with two circular arcs of radii 2 and 8 and opening angles of $5\pi/3$ and $\pi/3$, respectively, and with an additional point inside; and finally an equilateral triangle $\Delta = q_1q_2q_3$ with side length 10. The optimal matching between C and K and between K and Δ is shown in the lower part of Figure 5. It is obtained by superimposing the points o and p (marked by crosses) or the points p and q_1, respectively. The dimensions given in the lower middle part of Figure 5 also exhibit the position of the extra point in the keyhole K.

Let us first assume that the reference point $s(C)$ of the circle is its center o. Since $\delta_H(C, K) = 3$, the reference point $s(K)$ of K must lie in a circle of radius $3c$ around p. Similarly, $s(\Delta)$ must lie in a circle of radius $2c$ around $s(K)$, i.e., in a circle of radius $5c$ around q_1. If we turn K by 120° in both ways, we can conclude in a similar way that $s(\Delta)$ must lie in circles of radius $5c$ around q_2 and around q_3. By equivariance with respect to translations, the triangle Δ has only a single reference point, and therefore the three circles must intersect, as shown in the lower right part of Figure 5. This means that c must be at least $\sqrt{4/3} \approx 1.155$.

If the reference point $s(C)$ is not the center o, the only difference is that the centers of the final three circles will be translated by the respective amount. □

The above construction can be generalized to d dimensions to give a lower bound of $\sqrt{2d/(d+1)}$ on the quality c.

Comparing $\sqrt{4/3} \approx 1.155$ with the upper bound of $4/\pi \approx 1.27$ for the Steiner point, we note that the gap is quite small. We are quite sure that the lower bound can be improved, and we believe that the Steiner point is *the* optimal reference point with the best quality guarantee.

5 Acknowledgements

We would like to thank the following colleagues who in many discussions contributed ideas to this research: Franz Aurenhammer, Stefan Felsner, Michael Godau, Emo Welzl, Lorenz Wernisch, and Gerhard Wöginger.

References

[ABB] H. Alt, B. Behrends, J. Blömer, Approximate matching of polygonal shapes, *Proceedings 7th Annual Symposium on Computational Geometry*, 1991, pp. 186–193.

[AG] H. Alt, M. Godau, Computing the Hausdorff-distance in Higher Dimensions, in preparation.

[AST] P. J. Agarwal, M. Sharir, S. Toledo, Applications of parametric searching in geometric optimization, *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, 1992, pp. 72–82.

[B] B. Behrends, "Algorithmen zur Erkennung der ϵ-Kongruenz von Punktmengen und Polygonen", Diplomarbeit, FB Mathematik, Freie Universität Berlin, 1990.

[CGHKKK] P. P. Chew, M. T. Goodrich, D. P. Huttenlocher, K. Kedem, J. M. Kleinberg, D. Kravets, Geometric pattern matching under Euclidean motion, *Proc. 5th Canadian Conference on Computational Geometry*, 1993, pp. 151–156.

[CKS] P. P. Chew, K. Kedem, S. Schirra, On chracteristic points and approximate decision algorithms for Hausdorff distance, Report, Max-Planck-Institut für Informatik, Saarbrücken, 1993.

[G] B. Grünbaum, *Convex Polytopes*, Wiley & Sons, 1967.

[H] P. J. Heffernan, Generalized approximate algorithms for point set congruence, *Proc. 3rd Workshop on Algorithms and Data Structures*, Springer Lecture Notes in Computer Science, Vol. 709, 1993, pp. 373–384.

[HS] P. J. Heffernan, S. Schirra, Approximate decision algorithms for point set congruence, *Proc. 8th Annu. Symp. on Computational Geometry*, 1992, pp. 93–101.

[HK] D. P. Huttenlocher, K. Kedem, Computing the Minimum Hausdorff distance for point sets under translation, *Proc. 6th Annual Symp. on Computational Geometry*, 1990, pp. 340–349.

[HKS] D. P. Huttenlocher, K. Kedem, M. Sharir, The upper envelope of Voronoi surfaces and its applications, *Discrete and Computational Geometry*, **9** (1993), 267–291.

[HKR] D. P. Huttenlocher, G. A. Klanderman, W. J. Rucklidge, Comparing images using the Hausdorff-distance, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **15** (1993), 850–863.

[S] G. C. Shephard, The Steiner point of a convex polytope, *Canadian J. Math.*, **18** (1966), 1294–1300.

[Sch] R. Schneider, Krümmungsschwerpunkte konvexer Körper, *Abh. Math. Sem. Hamburg* **37** (1972), 112–132.

Piecewise-Linear Interpolation between Polygonal Slices*

Gill Barequet[†] Micha Sharir[‡]

Abstract

In this paper we present a new technique for piecewise-linear surface reconstruction from a series of parallel polygonal cross-sections. This is an important problem in medical imaging, surface reconstruction from topographic data, and other applications. We reduce the problem, as in most previous works, to a series of problems of piecewise-linear interpolation between each pair of successive slices. Our algorithm uses a partial curve matching technique for matching parts of the contours, an optimal triangulation of 3-D polygons for resolving the unmatched parts, and a minimum spanning tree heuristic for interpolating between non simply connected regions. Unlike previous attempts at solving this problem, our algorithm seems to handle successfully any kind of data. It allows multiple contours in each slice, with any hierarchy of contour nesting, and avoids the introduction of counter-intuitive bridges between contours, proposed in some earlier papers to handle interpolation between multiply connected regions. Experimental results on various complex examples, involving actual medical imaging data, are presented, and show the good and robust performance of our algorithm.

Keywords: surface reconstruction, tiling, surface fitting, branching surfaces, slice interpolation, triangulation, dynamic programming, geometric hashing, curve matching, polyhedra.

1 Introduction

The problem of reconstructing the boundary of a solid object from a series of parallel planar cross-sections has attracted much attention in the literature during the past two decades. The main motivation for this problem comes from medical imaging applications, where cross-sections of human organs, such as bones, tumors and tissues, are obtained by CT (Computerized Tomography) or MRI (Magnetic Resonance Imaging) apparata. These cross-sections, hereafter called *slices*, are the basis for interpolating the boundary surface of the organ. The interpolated object can then be displayed in graphics applications, or (more recently) even manufactured by an NC (Numerically Controlled) or an RP (Rapid Prototyping) machine. Another motivation for this problem is the non-destructive digitization of objects: after an object is scanned by an echo-graphic or an X-ray apparatus, the obtained slices are used for the reconstruction of the original object. Yet another motivation is the reconstruction of a 3-dimensional model of a terrain from topographic elevation contours.

Many solutions were suggested for the pure raster interpolation. These usually handle two raster images, where each pixel is either white or black, or assigned a grey-level taken from a fixed range. The interpolation produces one or more intermediate raster images, which smoothly and locally turn the first image into the second one. Then, the bounding surface is detected using other methods, such as edge detection techniques, for identifying places of transition from the inside to the outside of the object. In the grey level case, these methods include some thresholding mechanism which decides which levels are 'inside' the object and which are not. Cline et al. [6] attempted to convert directly the voxel data into a polyhedral surface, suggesting the *marching cubes* technique, which produced very small triangles whose size was roughly the same as that of the input voxels.

Many other solutions, including the approach taken in this paper, assume that the interpolation is preceded by an edge-detection process, which is invoked for each of the slices. Thus, each slice is then assumed to be represented by a hierarchy of non-crossing contours, each being a closed simple Jordan curve, which represent the boundaries between "material" and "non-material" areas; in general, the depth and breadth of this hierarchy is not restricted, and a contour may enclose any number of other contours, which themselves may enclose other contours, and so on. In practice, each contour is given as a discrete circular sequence of points along it, and we can thus regard it as a simple closed polygonal curve, whose vertices are the given points. Finally, we may also assume that the exterior, unbounded region in each planar slice represents "non-material" (the model is assumed to be bounded).

Thus the problem that we face is: Given a series of parallel planar slices (which we will assume to be parallel to the xy-plane), each consisting of a collection of non-crossing, but possibly nested, closed and simple polygonal curves, with the above properties, we want to reconstruct a polyhedral solid model whose cross sections along the given planes coincide with the input slices. A natural simplification of the problem, also taken in most earlier works, is to consider only a single pair of successive parallel slices, and to construct a solid model within the layer delimited by the planes of the slices, which interpolates between the given slices. The union (or, rather, concatenation) of these models will give us a solution model for the full problem.

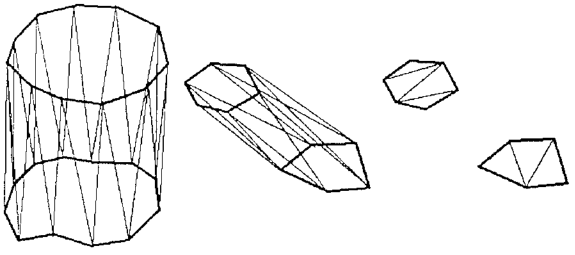
Before continuing, we should remark that the solution is not uniquely defined, and the measure of 'goodness'

*Work on this paper by both authors has been supported by a grant from the G.I.F., the German-Israeli Foundation for Scientific Research and Development. Work on this paper by the second author has also been supported by National Science Foundation Grant CCR-91-22103, and by grants from the U.S.-Israeli Binational Science Foundation, and the Israel Science Fund administered by the Israeli Academy of Sciences.

[†]School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel, and Algotec Systems Ltd., Raanana, Israel.

[‡]School of Mathematical Sciences, Tel-Aviv University, Tel-Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50



(a) Wrap-around tiling (b) Forced contour association (c) Separate treatment of non-overlapping contours

Figure 1: Contour association and tiling

of a proposed solution is rather subjective and intuitive. Of course, if each of the two slices consists of a single contour, and these contours roughly 'sit above each other', then we expect the solution to be a single 'drum-like' polytope whose boundary consists of a circular sequence of triangles 'wrapping around' the two contours; see Figure 1(a). However, even in the simple case of one contour in each slice, if the xy-projections of the two contours are far away from each other, it is not clear which is a better solution: to construct a highly slanted 'pipe' that connects between these contours (as in Figure 1(b)), or to regard the lower contour as the top cover of some pillar-like solid, and the upper contour as the bottom cover of another pillar-like solid (as in Figure 1(c)). The choice of a solution can become much more arbitrary in more involved cases.

We first briefly review the fairly extensive literature on this problem. Most of the earlier works only studied the variant where each slice contains only one contour. In the sequel we denote it as the *one-to-one* case, as opposed to the *one-to-many* and the *many-to-many* cases. These studies either sought a global optimization of some objective function, or settled with a local tiling-advancing rule, after the tiling starting points at the two contours were somehow determined, e.g. the closest pair of vertices between the contours.

In the full version we review these works in more detail, but since they are all inferior to our solution, in that they cannot handle arbitrary data, we omit this review here. These works are by Keppel [19], Fuchs et al. [9], Sloan and Painter [28], Cook et al. [7], Christiansen and Sederberg [5], Shantz [25], Batnitzky et al. [2], Sloan and Hrechanyk [26], Ganapathy and Dennehy [10], and by Wang and Aggarwal [29]. A good survey on all the works cited so far is given by Sloan and Painter [27]. They decompose each method into its building blocks, and compared the various methods accordingly. They also describe a testbed for evaluating and comparing these techniques.

Zyda, Jones and Hogan [32] made an attempt to handle the many-to-many case, but their method could not handle any branching cases, and produced unsatisfactory results for partially overlapping contours, as in Figure 4. Other limitations involved specific geometries. However, they suggested solutions for these limitations, which might help in certain cases, but which required some interaction with the user.

Boissonnat [3] presented a totally different approach. He constructed the Delaunay triangulation for each slice, projected one triangulation onto the other, and obtained a collection of tetrahedra, aiming to maximize the sum of their volumes. This was a considerable step towards handling the case where each slice has multiple contours. Boissonnat mentioned three typical examples where his standard method failed to produce good results, thus requiring special treatment. The first example contained two overlapping contours but with considerable differences in their geometry. The second example consisted of two similar contours, where one of them also contained a hole polygon. And the last example showed a branching problem but without contour overlaps. Our algorithm handles successfully (and easily) all these 'bad' examples. Boissonnat suggested a correction scheme, which either changed the geometry of one of the slices, or constructed one or two intermediate slices between the original ones. These idea were further developed by Boissonnat and Geiger ([4], [11]).

There have been a few other recent works that also attempted to handle the more general cases; these are by Kehtarnavaz and De Figueiredo [17], Kehtarnavaz, Simar and De Figueiredo [18], Ekoule, Peyrin and Odet [8], Meyers, Skinner and Sloan [22], and by Welzl and Wolfers [30]. However, all these treatments suffered from a variety of problems, treated only certain restricted cases, and worked only in favorable situations where the two slices closely resemble each other.

Finally, Gitlin, O'Rourke and Subramanian [12] prove that it is not always possible to find an interpolating polyhedron between two polygons, that lie in two parallel planes. That is, any attempted interpolation produces a self-intersecting surface. This result holds only when the interpolating triangles are all assumed to connect between the two polygons vertices, i.e. to have two vertices taken from one polygon and the third from the other polygon. We do not assume this in our approach, and we indeed get a non-intersecting interpolation when we apply our algorithm on their example.

Two comprehensive reviews of many of the works on reconstructing 3D objects from cross-sections are given by Schumaker [23] and by Hagen, Müller and Nielson [14]. The first review is not restricted to piecewise-linear constructions based on polygonal slices, but also refers to parametric and tensor representations of contours, and describes volumetric and surface approaches.

We propose a new approach to the interpolation problem. Our algorithm handles well slices with multiple contours, and does not rely on any resemblance between the slices. We accept slices which contain any number of contours, arbitrarily different in their geometries. The xy-projections of contours of different slices may arbitrarily overlap; we do not make any distinction in the treatment of contours which fully overlap, partially overlap or do not overlap at all. Many of the previous works, such as [19], [9], [26] and [10], either prohibit the creation of triangles in the same slice, or specifically define steps where this action is allowed. We do not make a distinction between triangles which connect the two slices and those which totally lie within a single slice. Generally speaking, our algorithm is based on an analysis of the resemblance between the two slices. Thus, we separately treat similar contour portions that are matched between the two slices, and then treat the remaining portions that do not match. We refrain from creating artificial bridges between contours that might conflict with the geometry of the other slice, as already noted in [25] and others. In the only case where we construct such bridges, they are guaranteed not to conflict with the geometry of the other slice, i.e. not to intersect the projection of any other contour. We do not have to introduce intermediate slices. To recap, our algorithm appears to overcome all the technical difficulties that hampered the performance of previous solutions, it treats data in full generality, and the extensive experimentation that we have conducted indicates that it performs very well on complicated large-size data. We

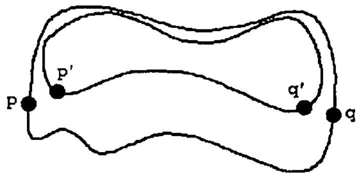


Figure 2: Matching contour portions

regard our algorithm as a significant step in the solution of this problem. For an illustration of the performance of our algorithm on real-life examples, see Figures 17 and 18.

Here is a brief overview of our algorithm. We first match similar contour portions between the two slices (e.g. the upper arcs pq and $p'q'$ in Figure 2). Then we 'stitch' (or 'tile') each pair of matched contour portions by a sequence of adjacent triangles forming a 'band' between the portions. With some care, if we take the union of the original contours and new bands, and cancel out duplicate edges, we obtain a collection of closed spatial polygonal curves, each of which may be composed of pieces of contours on both slices and of some edges of the connecting triangles. Moreover, our matching procedure essentially guarantees that the xy-projections of these curves are pairwise disjoint, although they may be nested within each other. If no nesting occurs, we simply triangulate each of these spatial polygons, using a simple dynamic programming approach, which roughly aims to minimize the total area of the triangulation. If nesting occurs, we take one polygon P with all polygons $P_1, \ldots, P_k$ whose xy-projections are directly nested within that of P, and apply a minimum spanning tree procedure that introduces edges connecting between these polygons and yielding an xy-projection which is simply connected, so we can then proceed to triangulate the resulting polygonal curve, as above. See Figure 3 for an illustration. More details of all these steps will be given later in the paper.

For the purpose of identifying matching portions of the contours we use a *partial curve matching* technique, which was first suggested by Kalvin et al. [16] and by Schwartz and Sharir [24]. This technique, which uses the so-called *Geometric Hashing* method, originally solved the following curve matching problem: Given two curves in the plane, such that one is a (slight deformation of a) proper subcurve of the other, find the translation and rotation of the subcurve that yields the best least-squares fit to the appropriate portion of the longer curve.

This technique was extended and used in computer vision for automatic identification of partially obscured objects in two or three dimensions, an important problem in robotics applications of computer vision, which has attracted much attention; see Hong and Wolfson [15], Wolfson [31], and Kishon, Hastie and Wolfson [20].

A simplified variant of this technique has recently been used by Barequet and Sharir [1] for the totally different problem of detecting and repairing gaps in the boundary of a polyhedron, a problem which often arises in the creation of polyhedral approximations of CAD models. In the variant used in [1], as well as the one used in this paper, no motion of one curve relative to the other is allowed, so the technique becomes considerably simpler.

The paper is organized as follows. In Section 2 we give a more precise definition of the problem and present an overview of the algorithm (more detailed than the one given above). The later sections describe in detail certain phases of the algorithm. Section 3 describes the matching of contour portions, and Section 4 describes the actual surface reconstruction. In Section 5 we briefly analyze the complexity of the algorithm, and present experimental results. We end in Section 6 with some concluding remarks.

2 Overview of the Algorithm

The input to the algorithm, as described in the Introduction, is a pair of parallel planar slices parallel to the xy-plane, each slice consisting of a list of closed and simple polygonal contours, which do not intersect each other. The containment hierarchy of the contours may be omitted; in this case we compute it ourselves, using a simple line-sweep technique. Contours of even nesting level (starting at 0, which labels external contours) are such that their interior, in a sufficiently small neighborhood of the contour, is the "material", and contours of odd level are those whose interior, sufficiently near them, is the "non-material". We orient each contour so that, when viewing the contour from above, the material lies to the right of the contour (thus all even-level contours are oriented in the clockwise direction, when viewed from above, and all odd-level contours are oriented in the counterclockwise direction). If necessary, we re-orient the contours in these consistent directions. We remark that we need to compute the contour hierarchy only to obtain the consistent orientation of contours: the hierarchy itself is not used in the algorithm.

Our proposed algorithm consists of the following steps:

1. Data acquisition:

(a) Orient all the contours in each slice in consistent directions, as explained above. If the input does not include this information, compute the contour nesting hierarchy in each slice, and use it to obtain the desired orientations.

2. Matching contour portions:

(a) Discretize each contour polygon into a cyclic sequence of vertices, so that the arc length between each pair of consecutive vertices is equal to some (small) given parameter.

(b) Vote for contour matches. Each pair of distinct vertices of the discretized contours, one on each slice, whose mutual horizontal distance is below some threshold parameter, contributes one vote. The vote is for the match between these two contours with the appropriate shift, which maps one of these vertices to the other.

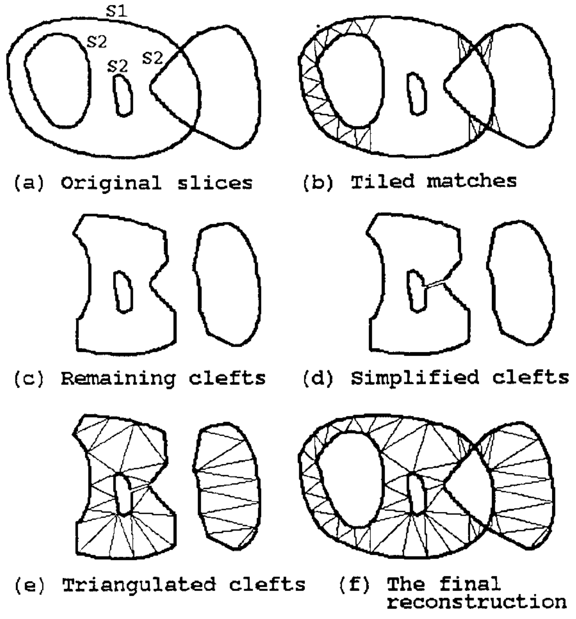
(c) Transform the resulting votes into a collection of candidates of partial contour matches.

3. Reconstructing the surface:

(a) Stitch together each pair of contour portions that have been matched in the above step, by adding triangles which connect between these portions. The new triangles are oriented consistently with the contours.

(b) Combine the remaining contour edges into spatial cycles (denoted as *clefts*), obtained by taking the union of the contour edges in both slices and of the edges of the stitching triangles, and by canceling out duplicate, oppositely-oriented edges. When projected onto the xy-plane, these cycles do not intersect each other. Find their nesting hierarchy, and, for each cleft C, construct, if necessary, a system of straight 'bridges' that connect between C and its holes (immediate children in the hierarchy), so as to turn them into a single cycle, which now replaces the cleft C.

(c) Triangulate the resulting 3-D clefts, using a 3-D min-



(a) Original slices (b) Tiled matches

(c) Remaining clefts (d) Simplified clefts

(e) Triangulated clefts (f) The final reconstruction

Figure 3: The different steps of our algorithm

imum area triangulation technique.

The various steps of the algorithm are illustrated in Figure 3. Figure 3(a) shows a pair of slices in a branching situation. The lower slice contains one contour (denoted by S_1), and the upper slice contains three contours (denoted by S_2). Figure 3(b) shows the tiling of the three matches found between these two slices. Figure 3(c) shows the remaining clefts, which form a shallow hierarchy of nested polygons. Figure 3(d) shows the clefts after the hole elimination step, and their minimum-area triangulations are shown in Figure 3(e). The final surface reconstruction is shown in Figure 3(f).

3 Matching Contour Portions

For lack of space, we omit here a detailed description of the data acquisition step, which was already briefly described above, and is anyway fairly straightforward.

3.1 Contour Discretization

Each contour polygon is refined and discretized into a cyclic sequence of points. This is done by choosing some sufficiently small arc length parameter s, and by generating equally-spaced points, at distance s apart from each other (along the polygon boundary). We need to choose s so that it is much smaller than (a) the length of any original edge of any contour, and (b) the minimum straight distance between any pair of contour points which lie on different contours (on the same slice) or lie on the same contour and their distance along the contour is sufficiently large.

3.2 Voting for Contour Matches

We first try to match pairs of portions of contours, where each portion in a pair belongs to a different slice. These matches aim to detect regions of similarity between the boundaries of the interpolating object along the two slices. Naturally, contour portions which are similar in the two slices must have similar sequences of footprints. Thus, our next goal is to search for pairs of sufficiently long subsequences that closely match each other. In our approach, two subsequences $(p_i, \ldots, p_{i+\ell-1})$ and $(q_j, \ldots, q_{j+\ell-1})$ are said to closely match each other, if, for some chosen parameter $\varepsilon > 0$. the number of indices k for which $\|p_{i+k} - q_{j+k}\| \leq \varepsilon$ is sufficiently close to ℓ. (Here the norm $\|\cdot\|$ is the Euclidean distance between the xy-projections of the points.) We perform the following voting process, where votes are given to good point-to-point matches.

The contours are given as cyclic ordered sequences of vertices, and we break each of them arbitrarily to make it into a linear sequence. All the vertices of the lower slice are preprocessed for *range-searching*, so that, for each vertex v of the upper slice, we can efficiently locate all the vertices of the lower slice that lie in some ε-neighborhood of (the xy-projections of) v. We have used a simple heuristic projection method for the range searching, that reduces the search to a pair of 1-dimensional searches; this may be inefficient in the worst case, but works very well in practice.

The positions along a contour sequence c, whose length is ℓ_c, are numbered from 0 to $\ell_c - 1$. Assume that the querying vertex v is in position i of contour sequence c_1. Then, each vertex retrieved by the query, which is in position j in contour sequence c_2, contributes a vote for the match between contours c_1 and c_2 with a shift equal to $(j - i) \pmod{\ell_{c_2}}$.

Obviously, matches between long portions of contours are reflected by a large number of votes for the appropriate shift between the matching contours. We thus collect the shifts with a large number of votes, identify (roughly) the starting and ending points of the corresponding match, and possibly also remove (small) portions of the matches, near their endpoints, which are also shared by other matches.

Note that this procedure will only match contour portions that are consistently oriented. This is important, since it guarantees that the material region lies on the same side of the two contours, thus implying that if we stitch the gap between these two contours, we get a proper portion of the boundary of a possible interpolating solid. Also note that the ε parameter for the range-searching queries is not a function of the input. It is rather our *a priori* estimation of the physical size of the difference between similar contour portions of the two slices.

In some cases small portions of the contours are included in more than one candidate match. This usually happens at the connection between two different matches, involving the same contour on one slice and different contours on the other. We simply eliminate those portions common to more than one significant candidate match.

3.3 Accepting Match Candidates

Each match is given a *score*. The score may also reflect, in addition to the number of votes for the appropriate shift, other quality measures (such as the closeness of the vertices on the two matching contour portions). Our setting of the scoring function is described in Section 5.

A crucial property of the matches, which we have to achieve when tuning the parameters which control the detection of matches, is that every intersection between the xy-projections of any pair of contours, one from each slice, will lead to a match between the corresponding

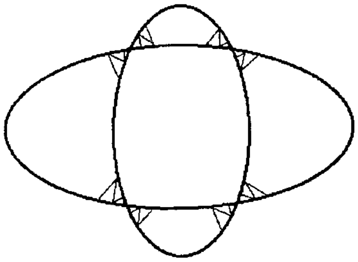


Figure 4: Intersecting contours with no long matching portions

contours, although it might be very short. This will ensure that there will be at least one match between each pair of overlapping contours (namely, contours with intersecting xy-projections), even if the relative amount of overlap is very small. The reason for this is given shortly below.

We achieve this by accepting very short match candidates (even of length 2 or 3), as long as their quality fits our bounds. We have to make sure that the discretization parameter is sufficiently small with respect to the voting threshold and with respect to the smallest contour feature size, as described above, in order not to miss contour intersections. An extreme example is shown in Figure 4, where four short matches are indeed found between the two contours.

4 Reconstructing the Surface

4.1 Stitching the Matches

Each match consists of two directed polygonal chains, whose xy-projections are very close to each other. We arbitrarily choose one end of the match, and 'merge' the two chains as if they were sorted lists of numbers. In each step of the merge we have pointers to the current vertices, u and v, in the two chains, and make a decision as to which chain should be advanced, say v advances to a new vertex w. Then we add the new triangle $\triangle uvw$ to the boundary of the constructed polyhedron, and advance the current vertex (from v to w) along the appropriate chain. When we reach the last vertex of one chain, we may further advance only the other one. This process terminates when we reach the last vertices of both chains.

The triangles that we create are oriented consistently with the contours. To achieve this, we invert the orientation of all contour polygons of one slice, say the lower one, and orient each triangle so that the edge it shares with a contour is oriented oppositely to the contour; see Figure 5 for an illustration of the directions of the tiling triangles. This orientation guarantees that the reconstructed solid boundary is oriented in a consistent manner.

Several advancing rules were examined, and the following simplest one proved itself the best. Assume that the current vertices of the two chains are v_i^1 and v_j^2, which are followed by v_{i+1}^1 and v_{j+1}^2, respectively. Then, if $|v_i^1 v_{i+1}^1| + |v_j^2 v_{i+1}^1| < |v_i^1 v_{j+1}^2| + |v_j^2 v_{j+1}^2|$, we advance the first chain; otherwise we advance the second chain[1]. This bears close resemblance to the merging of two sorted lists, and turns out to produce reasonably looking triangulated boundary patches between the matched contour portions. Figure 5 shows such a triangulation. Alternative advancing rules were described by Christiansen and Sederberg [5], and by Ganapathy and Dennehy [10].

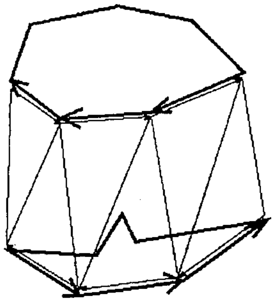
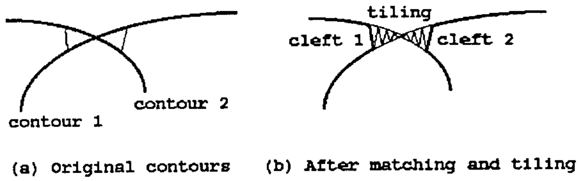


Figure 5: Tiling a match



(a) Original contours (b) After matching and tiling

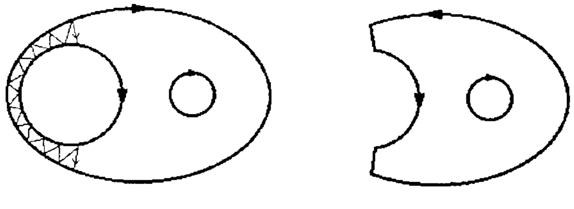
Figure 6: A short match and the clefts near an intersection of contours

4.2 Filling the Clefts

After tiling the matching contour portions. we remain with the unmatched portions. These, combined with the extreme edges of the tiling sequences, form a collection of closed 3-D polygons, which we refer to as *clefts*. Finding these clefts is straightforward. First. recall that we have already inverted the orientation of all contour polygons of the lower slice. Consider the contour polygons of the two slices (the inverted lower slice and the upper slice), as well as the tiling triangles. as the formal sum of their directed edges, and add up all these polygons, with the convention that $\vec{e} + (-\vec{e}) = 0$. After these cancellations, the resulting sum consists of all the desired clefts. Since each polygon that participates in the formal sum is a directed cycle, the resulting sum is easily seen to represent a collection of pairwise edge-disjoint directed cycles.

This collection of 3-D polygonal cleft cycles has the property that, when projected onto the xy-plane, no two cycles intersect. This is because we have already detected all the projected contour intersections in the matching phase, and because the extreme tiling edges in the triangulation of a match are likely to degenerate or almost degenerate in the projection. This situation is illustrated in Figure 6. The only way this property can be violated is when an extreme tiling edge crosses another edge of a contour. For this to happen, either the extreme edge must be highly slanted, or the contour must have very sharp turns. This can be avoided by an appropriate fine tuning of the discretization and matching threshold parameters, and in any case we did not face such a situation in all of our comprehensive experiments (which involved fairly complex and rather 'adversary' data). To recap, while in rare, worst-case

[1] That is, we advance so that the newly added triangle has smaller perimeter; actually, for program efficiency, we eventually used the squares of the distances, with equally good results.



(a) Original contour orientations

(b) Cleft cycle orientations

Figure 7: Cleft orientations

scenarios, projected cleft cycles might intersect, we will assume in what follows that this does not occur.

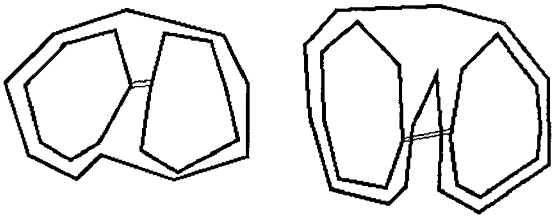
The xy-projections of these cleft polygons might again form a hierarchy of polygon nesting. We check this possibility by invoking again the same line-sweeping procedure used in the data acquisition step. This time, all the polygons are guaranteed to be correctly oriented. Figure 7 illustrates this situation. Figure 7(a) shows a branching case, where the lower slice contains one contour, and the upper slice contains two contours. After tiling the detected match, inverting the lower slice, and canceling out opposite edge occurrences, two nested cleft cycles remain, already oriented consistently, as shown as Figure 7(b).

Let C be a cleft whose immediate children in the hierarchy are $C_1, \ldots, C_k$. We define an undirected weighted complete graph G, so that each vertex of G is one of these $k+1$ cycles, and the weight of an edge connecting two cycles is the minimum distance between the xy-projections of these cycles. (We assume that our contour discretization is dense enough, so that the minimum vertex-to-vertex distance, which is what we actually computed, serves as a sufficiently good approximation of the actual minimum distance.) We now compute a minimum spanning tree T of G, and form a bridge between each pair of cycles connected by an edge of T; the bridge is a straight segment connecting the two nearest vertices on these cycles. It is easily verified that the xy-projections of the bridges do not cross each other, and also do not cross the xy-projection of any cycle. We create two oppositely-oriented copies of each bridge and add them to the given cycles. This eliminates the 'windows' $C_1, \ldots, C_k$ and replaces the whole configuration by a single composite, self-touching but otherwise simple polygonal cycle.

We emphasize that bridge assembly was the most significant obstacle in previous works, which used this tool for reducing a branching situation to the simple one-to-one case. That was because a bridge in one slice could conflict with the geometry of the other slice, by having its projection intersect a contour, as shown in Figure 8(b). As noted, we do not face this problem.

We note that this procedure is required only in complicated cases (few of them are presented in Section 5); in most practical instances clefts do not tend to be nested. In any case, after this hole elimination step, we are left with a collection of closed polygonal cleft cycles, with the property that their xy-projections enclose pairwise-disjoint regions.

Our next goal is to triangulate each cleft cycle. Since many such triangulations are possible, we seek a triangulation which minimizes the total area of the triangles. More precisely, we want to solve the following problem: Given a 3-dimensional closed polygonal curve P, and an objective function $\mathcal{F}$ defined on all triangles (called *weight* in the sequel), find the triangulation of P (i.e., a



(a) An acceptable bridge

(b) A conflicting bridge

Figure 8: Bridges in simple branching cases

collection of triangles spanned by the vertices of P. so that each edge of P is incident to exactly one triangle, and all other triangle edges are incident to two triangles each), which minimizes the total sum of $\mathcal{F}$ over its triangles.

For this purpose, we closely follow the dynamic programming technique of Klincsek [21] for finding a polygon triangulation in the plane, which minimizes the total sum of edge lengths. Let $P = (v_0, v_1, \ldots. v_{n-1}, v_n = v_0)$ be the given polygon. Let $W_{i,j}$ $(0 \le i < j \le n-1)$ denote the weight of the best triangulation of the polygonal curve $(v_i, \ldots, v_j, v_{j+1} = v_i)$. We then apply a straightforward dynamic programming approach, which initializes each $W_{i,i+2}$ to $\mathcal{F}(v_i, v_{i+1}, v_{i+2})$. and then computes each $W(i,k)$, for $k - i = 3, 4, \ldots, n-1$. as $\min_{i<m<k}[W_{i,m} + W_{m,k} + \mathcal{F}(v_i, v_m, v_k)]$. We omit here the easy further details. The actual value of $\mathcal{F}(u, v. w)$ that we have used is a weighted average of several terms, the most significant one being the area of the triangle; see Section 5 for more details.

Finally, if the interpolation involves the uppermost or lowermost slice in the given sequence, we also have to add to the reconstructed object boundary the regions of material on that slice, so as to 'close' the volume of that object.

5 Experimental Results

We have implemented the whole algorithm on a Digital DECstation 5000/240 and on a Sun SparcStation II in C. We have experimented with the algorithm on several data files obtained by CT or MRI scanners, and obtained very good results in practically all cases. The input usually consisted of about fifty to one hundred cross-sections, from 0.5 mm to 2.0 mm apart. The tuning of the parameters (discretization length and size of neighborhood in the geometric hashing) was very robust, and large variation of these parameters produced nearly identical results. We usually used 1.0 mm as the discretization parameter, and 3.0 mm for the voting threshold (for human organs whose global size was between 5 to 20 cm in all dimensions). We allowed up to two successive point mismatches along a match. A point-to-point match contributed the amount of $1/(d + 0.1)$ to the match score, where d was the xy distance between the two points. We considered only match candidates which received 4 votes or more and whose scores were above 15.0. (Our discretization was sufficiently dense so that this choice still captured all contour overlaps.) The objective function $\mathcal{F}$ for the cleft-triangulation was taken to

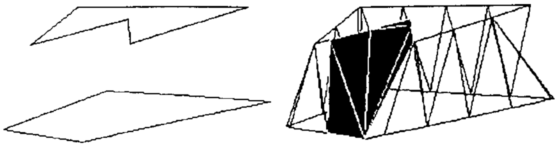


Figure 9: A synthetic example

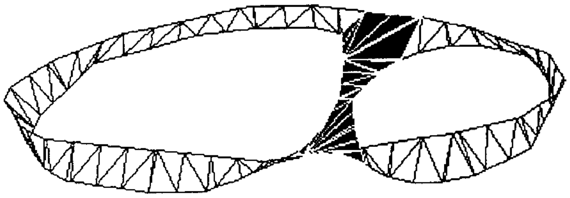


Figure 10: A synthetic branching example

be $0.85A + 0.05P + 0.10R$, where A is the area of the triangle, P is its perimeter, and R is the ratio between the largest and the smallest of its three edges. All these parameters were user defined, but modifying them did not achieve any better results.

We can measure the complexity of the algorithm in terms of four variables: k, the total number of input points along the contour edges of two consecutive slices, n, the total number of points after the arc length discretization step, ξ, the number of clefts, and h, the size of the biggest cleft (after the hole-elimination step). Usually, k is considerably smaller than n. The number ξ of clefts could be in the worst case as large as $\Theta(k^2)$, but in practice ξ is smaller than, or at least comparable with the number of contours c (which is typically much smaller than k).

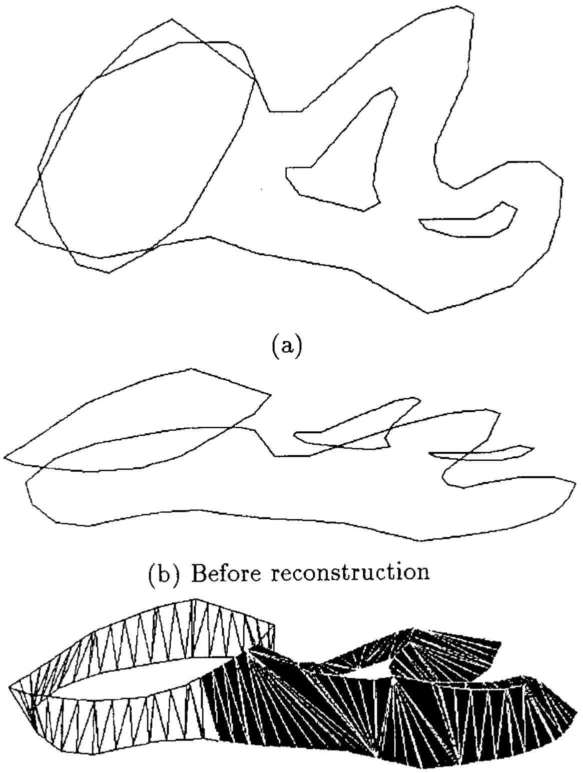
Due to lack of space, we do not give here a formal analysis of the complexity of the algorithm. Such an analysis produces terms that may be very large; for example, the cleft triangulation step (based on dynamic programming) takes $O(\xi h^3)$ time, which is, theoretically, and in complex situations also pragmatically, the most expensive portion of our algorithm. The entire algorithm runs on practical instances in average $O((k + \xi)\log(k + \xi) + n + \xi h^3)$ time.

Here are some specific examples of the performance of the algorithm:

Figure 9 shows a simple case, similar to the first problematic example of Boissonnat [3]. Each slice in this example contains exactly one contour. The tiled match appears in white, whereas the triangulated cleft appears in black.

Figure 10 shows a synthetic branching example, where the lower slice contains two contours, whereas the upper slice contains only one contour. Two long matches were found and tiled by the white triangles. The remaining cleft appears in between the two matches, and its triangulation appears in black.

Figure 11 shows a more complicated synthetic example, where the lower slice contains one contour, and the upper slice contains three contours. Figure 11(a) shows a top view of this situation, whereas Figure 11(b) shows an isometric view of it. Figure 11(c) shows the surface reconstruction. The tiling of the single match appears in white. The remaining cleft consists of one cycle which encloses two other cycles. Two bridge constructions compose the three polygons into a single one, and its triangulation appears in black.

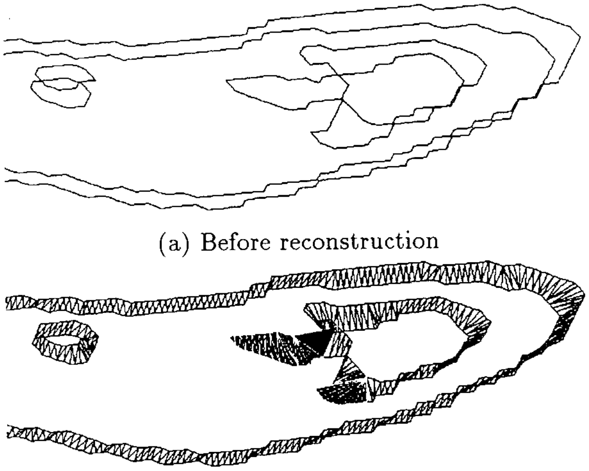


(a)

(b) Before reconstruction

(c) After reconstruction

Figure 11: A synthetic complicated example

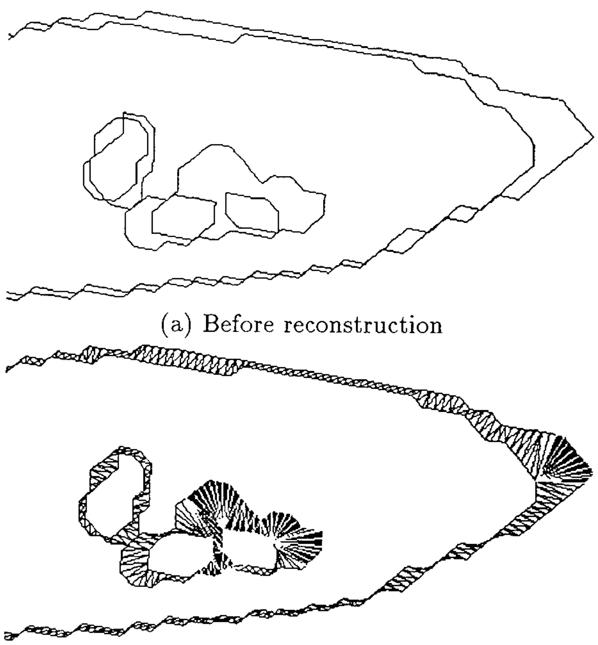


(a) Before reconstruction

(b) After reconstruction

Figure 12: A simple case

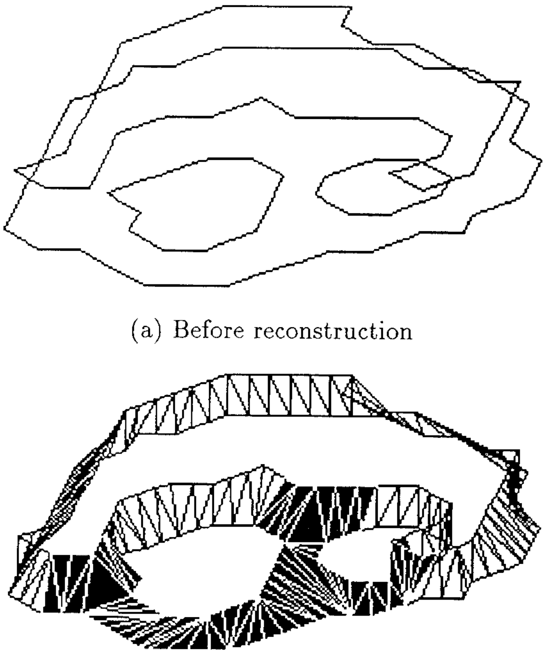
The next several figures show the typical performance of our algorithm, as observed from its execution on a series of cross-sections of a human jaw bone. Figure 12 shows the reconstructed surface between two slices. The tiles of the matches appear in white, and the triangulations of the two remaining clefts appear in black. Figure 13 shows two slices with a branching region. The match tiles appear in white, whereas the cleft triangulations appear in black. Figure 14 presents a consider-



(a) Before reconstruction

(b) After reconstruction

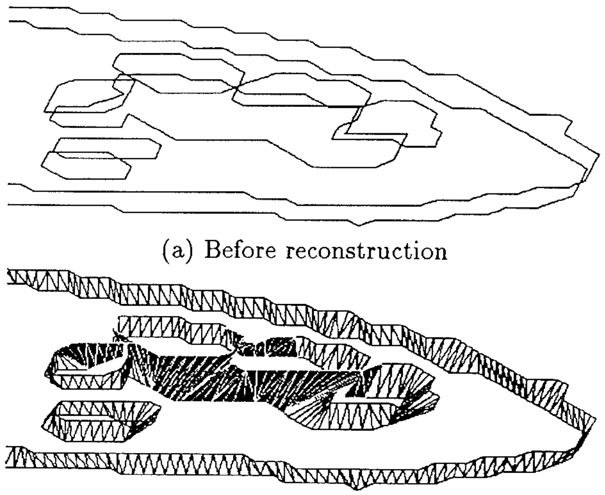
Figure 13: A simple branching case



(a) Before reconstruction

(b) After reconstruction

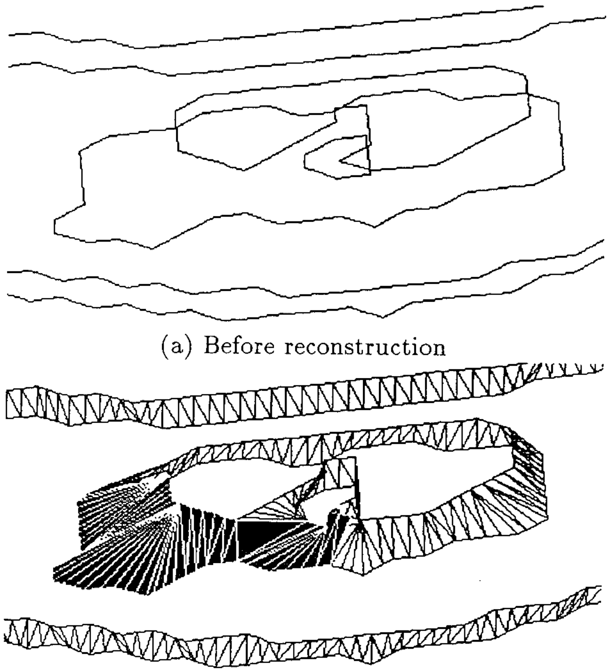
Figure 14: A complicated branching case



(a) Before reconstruction

(b) After reconstruction

Figure 15: A multiple branching case



(a) Before reconstruction

(b) After reconstruction

Figure 16: A composite case

ably more complicated situation, where the reconstruction turns out to be 'intuitively correct'. Figure 14(a) shows that the lower slice contains a contour with two hole contours, and the upper slice contains only one contour, which mostly lies above the "material" region of the lower slice. The surface reconstruction is shown in Figure 14(b). The three match tiles appear in white, and the cleft triangulations appear in black. The reader may verify that the two "non-material pillars" represented by the two hole contours in the lower slice were connected to the unbounded "non-material" region in the upper slice. Similar complex examples are given in Figures 15 and 16.

The reconstruction of the whole jaw bone, whose in-

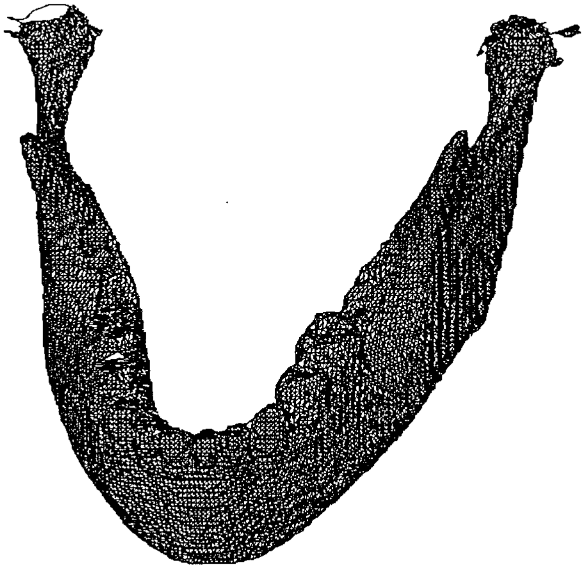


Figure 17: A fully reconstructed human jaw bone

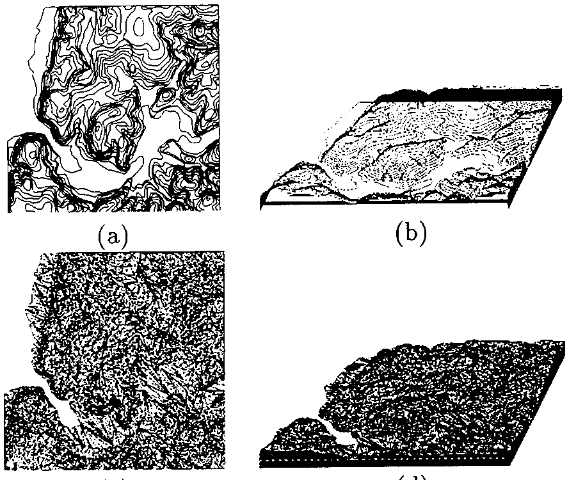


Figure 18: A topographic terrain

Model	Synthetic	Jaw Bone
Slices	2	96
Layers	1	95
Contours:		
Total	3.0	417.0
Per Slice	1.5	4.3
Vertices:		
Total	143.0	17,646.0
Per Contour	47.7	42.3
Matches:		
Total	2.0	330.0
Per Layer	2.0	3.5
Clefts:		
Total	1.0	275.0
Per Layer	1.0	2.9
Time (Seconds):		
All:		
Stitching	0.07	12.50
Triangulation	0.13	37.12
Total	0.20	49.62
Per Layer	0.20	0.52

Table 1: Performance of the algorithm

put consisted of 96 slices, is shown in Figure 17. The result was a valid polyhedral description, which contained about 60,000 triangles. The reconstructed jaw contained 209 3-D cavities (fully enclosed in the outer 3-D reconstructed boundary).

Our experimentations were mostly performed on medical imaging data. However, as we mentioned in the introduction, the reconstruction problem has other applications as well. One such application is the reconstruction of a terrain from elevation contour data in topographic maps. This application is much simpler, because of the xy-monotonicity of terrains. For example, contours cannot overlap in this case (although they can be nested). Figure 18 shows the reconstruction of a terrain in the Zikhron-Ya'akov area in Israel. Figures 18(a,b) show the elevation contours (a top view and an isometric view). The data contained seventeen levels which are 10 meters apart, starting from 20 and ending at 180 meters above see level. These levels consisted of 111 contours made of 5,329 edges. Figures 18(c,d) show the full reconstruction of the terrain (again, a top view and an isometric view).

Finally, we measured the performance of our implementation on two of the examples described above. All the time measurements were taken on a Digital DECstation 5000/240. Table 1 summarizes the performance of the algorithm on the simple branching case shown in Figure 10, and on the whole jaw bone shown in Figure 17. Note that the data of the jaw bone was accumulated from 95 layer reconstructions. Therefore, some of the data is averaged per layer. We note that in medical data, at least the data with which we experimented, successive slices tend to differ a lot in their geometries. Therefore, many clefts are created and their (relatively time-consuming) triangulations affect the total running time of the algorithm.

6 Conclusion

We have proposed in this paper an algorithm for solving the practical problem of polyhedral interpolation between parallel polygonal slices. This problem has many medical and geographic applications, and also appears to be a fairly basic and interesting problem in computer graphics and solid modeling.

Our method produces a relatively smooth boundary, due to the contour discretization. In situations where the addition of new vertices is not desired, e.g. due to data explosion, our system uses the discretization only for the matching step, but the tiling itself and the following minimum-area triangulation are performed on the contours containing only the original points.

We feel that our technique reconstructed the boundary of various organs in an intuitively appealing manner; one might say that our algorithm demonstrated some 'understanding' of the underlying problem. The results were more than adequate even in extreme cases of tiling between two seemingly totally different slices.

We plan to continue the experimentation with our algorithm, to test its performance limits and see if there are data instances on which the algorithm might not perform well, thus requiring further calibrations and enhancements (in view of our experimentation so far, we doubt that anything really problematic will arise).

Acknowledgment

We wish to thank Haim Wolfson for helpful discussions concerning the geometric hashing technique, Emo Welzl and Barbara Wolfers for helpful discussions concerning the solid reconstruction problem, and for providing us with data files and with a comprehensive bibliography on that problem, and Jean-Daniel Boissonnat and Bernhard Geiger for supplying us with data files.

References

[1] G. Barequet and M. Sharir, Filling gaps in the boundary of a polyhedron, Technical Report 277/93, Department of Computer Science, Tel Aviv University, 1993.

[2] S. Batnitzky, H.I. Price, P.N. Cook, L.T. Cook and S.J. Dwyer III, Three-dimensional computer reconstruction from surface contours for head CT examinations, *J. of Computer Assisted Tomography*, 5 (1981), 60–67.

[3] J.D. Boissonnat, Shape reconstruction from planar cross sections, *Computer Vision, Graphics and Image Processing*, 44 (1988), 1–29.

[4] J.D. Boissonnat and B. Geiger, Three dimensional reconstruction of complex shapes based on the Delaunay triangulation, Technical Report 1697, Inria-Sophia Antipolis, 1992.

[5] H.N. Christiansen and T.W. Sederberg, Conversion of complex contour line definitions into polygonal element mosaics, *Computer Graphics*, 13 (1978), 187–192.

[6] H.E. Cline, W.E. Lorensen, S. Ludke, C.R. Crawford and B.C. Teeter, Two algorithms for the three-dimensional reconstruction of tomograms, *Medical Physics*, 15 (1988), 320–327.

[7] L.T. Cook, P.N. Cook, K.R. Lee, S. Batnitzky, B.Y.S. Wong, S.L. Fritz, J. Ophir, S.J. Dwyer III, L.R. Bigongiari and A.W. Tempelton, An algorithm for volume estimation based on polyhedral approximation, *IEEE Transactions on Biomedical Engineering*, 27 (1980), 493–500.

[8] A.B. Ekoule, F.C. Peyrin and C.L. Odet, A triangulation algorithm from arbitrary shaped multiple planar contours, *ACM Transactions on Graphics*, 10 (1991), 182–199.

[9] H. Fuchs, Z.M. Kedem and S.P. Uselton, Optimal surface reconstruction from planar contours, *Communications of the ACM*, 20 (1977), 693–702.

[10] S. Ganapathy and T.G. Dennehy, A new general triangulation method for planar contours, *ACM Transactions on Computer Graphics*, 16 (1982), 69–75.

[11] B. Geiger, Construction et utilisation des modèles d'organes en vue de l'assistance au diagnostic et aux interventions chirurgicals, Ph.D. Dissertation, L'Ecole des Mines de Paris, 1993.

[12] C. Gitlin, J. O'Rourke and V. Subramanian, On reconstructing polyhedra from parallel slices, Technical Report 025, Department of Computer Science, Smith College, Northampton, MA, 1993.

[13] L. Guibas and J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *ACM Transactions on Graphics*, 4 (1985), 74–123.

[14] H. Hagen, H. Müller and G.M. Nielson, *Focus on Scientific Visualization*, Springer Verlag, 1993.

[15] J. Hong and H.J. Wolfson, An improved model-based matching method using footprints, *Proc. 9th Int. Conf. on Pattern Recognition*, 1988, 72–78.

[16] A. Kalvin, E. Schonberg, J.T. Schwartz and M. Sharir, Two-dimensional, model-based, boundary matching using footprints, *Int. J. of Robotics Research*, 5 (1986), 38–55.

[17] N. Kehtarnavaz and R.J.P. De Figueiredo, A framework for surface reconstruction from 3D contours, *Computer Vision, Graphics and Image Processing*, 42 (1988), 32–47.

[18] N. Kehtarnavaz, L.R. Simar and R.J.P. De Figueiredo, A syntactic/semantic technique for surface reconstruction from cross-sectional contours, *Computer Vision, Graphics and Image Processing*, 42 (1988), 399–409.

[19] E. Keppel, Approximating complex surfaces by triangulation of contour lines, *IBM Journal of Research and Development*, 19 (1975), 2–11.

[20] E. Kishon, T. Hastie and H. Wolfson, 3-D curve matching using splines, *J. of Robotic systems*, 8 (1991), 723–743.

[21] G.T. Klincsek, Minimal triangulations of polygonal domains, *Annals of Discrete Mathematics*, 9 (1980), 121–123.

[22] D. Meyers, S. Skinner and K. Sloan, Surfaces from contours: the correspondence and branching problems, *Proc. Graphics Interface '91*, 1991, 246–254.

[23] L.L. Schumaker, Reconstructing 3D objects from cross-sections, in: W. Dahmen, M. Gasca and C.A. Micchelli (eds.), *Computation of Curves and Surfaces*, Kluwer Academic Publishers, 1989, 275–309.

[24] J.T. Schwartz and M. Sharir, Identification of partially obscured objects in two and three dimensions by matching noisy characteristic curves, *Int. J. of Robotics Research*, 6 (1987), 29–44.

[25] M. Shantz, Surface definition for branching contour-defined objects, *Computer Graphics*, 15 (1981), 242–270.

[26] K.R. Sloan and L.M. Hrechanyk, Surface reconstruction from sparse data, *Proc. IEEE Conf. on Pattern Recognition and Image Processing*, 1981, 45–48.

[27] K.R. Sloan and J. Painter, From contours to surfaces: testbed and initial results, *Proc. CHI + GI '87*, 1987, 115–120.

[28] K.R. Sloan and J. Painter, Pessimal guesses may be optimal: A counterintuitive search result, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10 (1988), 949–955.

[29] Y.F. Wang and J.K. Aggarwal, Surface reconstruction and representation of 3D scenes, *Pattern Recognition*, 19 (1986), 197–207.

[30] E. Welzl and B. Wolfers, Surface reconstruction between simple polygons via angle criteria, *Proc. 1st European Symp. on Algorithms*, 1993, 397–408.

[31] H.J. Wolfson, On curve matching, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12 (1990), 483–489.

[32] M.J. Zyda, A.R. Jones and P.G. Hogan, Surface construction from planar contours, *Computers and Graphics*, 11 (1987), 393–408.

Practical Methods for Approximate Geometric Pattern Matching under Rigid Motions
(Preliminary Version)

MICHAEL T. GOODRICH*
Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218, USA
E-mail: goodrich@cs.jhu.edu

JOSEPH S. B. MITCHELL†
Department of Applied Math. Statistics
SUNY, Stony Brook, University
Stony Brook, NY 11794, USA
E-mail: jsbm@ams.sunysb.edu

MARK W. ORLETSKY
Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218, USA
E-mail: orletsky@cs.jhu.edu

Abstract

We present practical methods for approximate geometric pattern matching in d-dimensions along with experimental data regarding the quality of matches and running times of these methods verses those of a branch-and-bound search. Our methods are faster than previous methods but still produce good matches.

1 Introduction

Suppose we are given a set B of n points in $\Re^d$, which we shall call the *background*, and a set P of m points in $\Re^d$, which we shall call the *pattern*. The *geometric pattern matching* problem is to determine a rigid motion, taken from some class of motions, such that each point in P is moved to a point in B. This problem is motivated by a number of problems in computer vision, including character recognition. Of course, it is easy to solve this problem in $O(nm \log n)$ time just by keeping the points of B in some dictionary.

Unfortunately, this approach is very sensitive to noise in the background. Thus, it is actually more natural to ask for a rigid motion of P such that each point of P is moved *near* to a point in B. We call this the *approximate geometric pattern matching* problem. Formally, we desire a rigid motion T, taken from some class of motions $\mathcal{T}$, such that the directed[1] Hausdorff distance between $T(P)$ and B is minimized. Recall that the directed Hausdorff distance, $h(C, D)$, from a point set C to another point set D is defined as

$$h(C, D) = \max_{c \in C} \min_{d \in D} \rho(c, d),$$

where ρ is the usual Euclidean distance between c and d.

1.1 Previous Work

This line of research has a rich history, which really begins in earnest in the computational geometry literature with work by Alt, Mehlhorn, Wagener, and Welzl [3] on methods for finding *congruences* between two sets of points A and B under rigid motions. Besides giving a number of exact methods, they introduce an important *approximate* version of the problem, where one is given a tolerance $\epsilon > 0$ and asked to find a motion T, if it exists, that allows a matching between each point in $T(A)$ and a point in B at most distance ϵ away. Moreover, they also consider the *optimization* version of this problem, in which one wishes the smallest ϵ admitting such a motion. This version of the problem is very close to the problem we address in this paper. Unfortunately, the running times for this version of the problem are quite high.

Imai, Sumino, and Imai [21] show that these bounds can be reduced somewhat if one is already given an assignment of points in A to points in B. Likewise, Arkin, Kedem, Mitchell, Sprinzak, and Werman [4] show that one can improve the running times in the approximate case if the "noise regions" are disjoint. Even so, the methods derived in these papers are still relatively nontrivial and the running times for all but the most simple motions are still quite high.

In work more directly related to this paper, several researchers [6, 7, 18, 19, 20, 25] have studied

*This research was partially supported by the NSF and DARPA under Grant CCR-8908092, by the NSF under Grants CCR-9003299, CDA-9015667, and IRI-9116843.

†This research was partially supported by grants from Hughes Research Laboratories, Boeing Computer Services, Air Force Office of Scientific Research contract AFOSR-91-0328, and by NSF Grants ECSE-8857642 and CCR-9204585.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

[1] The undirected Hausdorff distance is defined as $H(C, D) = \max\{h(C, D), h(D, C)\}$.

methods for finding rigid motions that minimize either the directed or undirected Hausdorff distance between the two point sets. All of these methods are based upon intersecting higher-degree curves and/or surfaces, which are then searched (sometimes parametrically [1, 8, 9, 10, 22]) to find a global minimum. This reliance upon intersection computations makes for algorithms that are potentially numerically unstable, are conceptually complex, and have running times that are high for all but the most trivial motions. Indeed, Rucklidge [26] gives evidence that such methods *must* have high running times.

The high running times of these methods motivated Heffernan [16] and Heffernan and Schirra [17] to consider an approximate decision problem for approximate point set congruence (they did not study point set pattern matching). The general framework is that one is given a parameter $\epsilon > 0$ and asked to solve the approximate set congruence problem [3], except that one is allowed to "give up" if one discovers that ϵ is "too close" to the optimal value ϵ^* (i.e., the Hausdorff distance between $T(A)$ and B). By allowing algorithms to be "lazy" in this way, they show that the running times can be significantly reduced from the previous bounds. However, this may not be the best way to approach the problem with an approximation algorithm, since in some instances the program will consume a possibly large amount of computing time and terminate without an approximation. Unfortunately, the running times of the methods in [16, 17] increase substantially if one tries to get a yes-or-no answer for an ϵ close to ϵ^*. Thus, it is difficult to use their methods to approximate ϵ^*.

1.2 Our Results

In this paper we present a simple (almost trivial) approach for approximate point set pattern matching under rigid motions, where one is given a pattern set P of m points in $\Re^d$ and a background B of n points in $\Re^d$ and asked to find a rigid motion T that minimizes $h(T(P), B)$. Our methods are based upon a simple "pinning" strategy. They are fast, easy to implement, and do not suffer from any numeric instability difficulties. Moreover, since they are defined for the (more general) directed Hausdorff measure, they are tolerant of noise in the background. Our methods are not exact, however.

Instead, in the spirit of approximation methods for other hard problems (e.g., NP-hard problems [11, 14]), we derive algorithms that are guaranteed to come close to the optimal value, δ^*, which is also the approach taken by Alt *et al.* [2] for approximate polygon matching. In particular, each of our methods gives a rigid motion T such that $h(T(P), B) \leq \alpha\delta^*$, for some small constant $\alpha > 1$.

Motion	*Best Match*	*Our Method*	*Factor*
$\mathcal{T}$ in $\Re^2$	$nm^2 \log n$ [19]	$nm \log n$	2
$\mathcal{TR}$ in $\Re^2$	$n^2m^3 \log n$ [6]	$n^2m \log n$	4
$\mathcal{T}$ in $\Re^d$	$n^3m^2 \log^2 n$ [19]	$nm \log n$	$2+\epsilon$
$\mathcal{R}$ in $\Re^3$	–	$n^2m \log n$	$4+\epsilon$
$\mathcal{TR}$ in $\Re^3$	–	$n^3m \log n$	$8+\epsilon$

Table 1: **Our Results.** We give the previous best-match asymptotic running times, our asymptotic running times, and our worst-case approximation factor guarantee. The transformations considered are $\mathcal{T}$ =translation and $\mathcal{R}$ =rotation, and their combination. The parameter ϵ can be any positive constant less than 1. Also, the previous methods for the 3-d translation case is actually for undirected Hausdorff distance, and the translation method in [19] is for $d \leq 3$.

We justify the implementability of our methods through empirical tests. In particular, we study the running time and match quality performance of our methods when run on various input instances, and compare this performance with that of a more conventional procedure based upon a branch-and-bound search of a discretized configuration space. Our experimentation gives evidence that our methods are indeed quite fast in practice, yet nevertheless are still able to find rigid motions with good matches. In addition to this analysis, we also give some heuristics that speed up the running time in practice, even though they do not actually improve the worst-case running time.

In the next section we describe our methods under various rigid motions, in 2-, 3-, and, in some cases, higher dimensions. Our results are summarized in Table 1. We give some of the results of our experimental work in Section 3, and we conclude in Section 4.

2 Our Algorithms

In this section we give each of our methods for approximate geometric pattern matching including bounds on the quality of match that will be produced relative to the absolute best possible match.

2.1 Translation Only in $\Re^d$

Suppose we are given a set B of n points in $\Re^d$ and a set P of m points in $\Re^d$, where d is considered constant. In this subsection we give an efficient method for finding a translation T' such that $h(T'(P), B)$ is at most $2h(T_{\text{opt}}(P), B)$, where $T_{\text{opt}}(P)$ is the *optimal* translation, i.e., the one that minimizes $h(T(P), B)$, taken over all translations T.

Pick some point $p \in P$ as a "representative" for P. For each $b \in B$, define T_b to be the translation that takes p to b. Our method, then, is to find $\min_{b \in B}\{h(T_b(P), B)\}$ as our best approximate match, and let T' be the translation T_b that achieves this bound. This can clearly be done in $O(nm * \mathit{Nearest}_d(n))$, where $\mathit{Nearest}_d(n)$ is the time needed to perform a nearest-neighbor query in an n-point set (in this case B) in $\Re^d$.

Of course, if $d = 2$, then we can achieve $\mathit{Nearest}_2(n) = O(\log n)$ by answering nearest-neighbor queries using point location in a Voronoi diagram for B (e.g., [12, 23, 24]), which requires $O(n \log n)$ preprocessing. For higher dimensions, this approach is not as efficient, however. So, our method for implementing nearest neighbors will instead be based on the (practical) method of Arya *et al.* [5], which finds *approximate* nearest neighbors in $O(\log n)$ time in $\Re^d$, for any constant dimension $d \geq 2$, after $O(n \log n)$ preprocessing. Their method can be tuned to return a point whose distance is at most a $(1+\epsilon)$-factor larger than the nearest-neighbor distance, for any constant $0 < \epsilon < 1$.

Lemma 2.1 *$h(T'(P), B) \leq 2h(T_{\text{opt}}(P), B)$ in $\Re^2$ and $h(T'(P), B) \leq (2+\epsilon)h(T_{\text{opt}}(P), B)$ in $\Re^d$, for $d \geq 3$, with $\epsilon > 0$ being any constant less than* 1.

Proof: (For $\Re^2$) For simplicity of expression, define $h_{\text{opt}} = h(T_{\text{opt}}(P), B)$. Observe that for each $p \in T_{\text{opt}}(P)$, there exists an associated $b \in B$ that is within a distance h_{opt} of p. Consider the process of translating the entire pattern $T_{\text{opt}}(P)$ so that a particular point p now coincides with its associated background point b. This translation will cover a distance of at most h_{opt} and will therefore increase the distance from any other point in the pattern to its associated background point by at most h_{opt}. Therefore, it will have a directed Hausdorff distance of at most twice that of $T_{\text{opt}}(P)$. This translation will be one of those generated and checked by our algorithm. Thus, our algorithm will produce a translation that has a directed Hausdorff distance that is at most a factor of two times the minimal. (For $\Re^d$, with $d \geq 3$, this factor is multiplied by $(1+\epsilon')$, where ϵ' is defined so that the approximation factor for our method becomes $2+\epsilon$.) ∎

2.2 Translation and Rotation in $\Re^2$

Suppose we are given a set B of n points in $\Re^2$ and a set P of m points in $\Re^2$. In this subsection we give an efficient method for finding a Euclidean motion (translation and rotation) E' such that $h(E'(P), B)$ is at most $4h(E_{\text{opt}}(P), B)$, where $E_{\text{opt}}(P)$ is an *optimal* Euclidean motion, i.e., one that minimizes $h(E(P), B)$, taken over all valid motions E.

Select from the pattern diametrically opposing points and call them r and k. Point r is treated as both the distinct representative of the pattern for the translation part of the transformation and it is treated as the center of rotation for the rotation part of the transformation. Specifically, for each $b \in B$, define T_b to be the translation that takes r to b. Also for a $b' \in B, b' \neq b$, define $R_{b'}$ to be the rotation that makes r, b', and k collinear. Let $E_{b,b'}$ be the Euclidean motion that is the combination of T_b and $R_{b'}$. Our method is to find $\min_{b,b' \in B}\{h(E_{b,b'}(P), B)\}$ as our best approximate match, and let E' be the Euclidean motion $E_{b,b'}$ that achieves this bound. This can clearly be done in $O(n^2 m * \mathit{Nearest}_3(n))$ time, which we can implement to be $O(n^2 m \log n)$.

Lemma 2.2 *$h(E'(P), B) \leq 4h(E_{\text{opt}}(P), B)$, for any constant $0 < \epsilon < 1$.*

Proof: For simplicity of expression, define $h_{\text{opt}} = h(E_{\text{opt}}(P), B)$. Observe that for each $p \in E_{\text{opt}}(P)$, there exists an associated $b \in B$ that is within a distance h_{opt} of p. Consider the process of translating the entire pattern $E_{\text{opt}}(P)$ so that the particular point r now coincides with its associated background point b. This translation will cover a distance of at most h_{opt} and will therefore increase the distance from any other point in the pattern to its associated background point by at most h_{opt}. Now consider the process of rotating the entire pattern about point r so that line between r and k now passes through the background point associated with k in $E_{\text{opt}}(P)$. This rotation will have the effect of moving point k by at most $2h_{\text{opt}}$. Since k is the fartherest point in the pattern from the center of rotation, all other pattern points will be moved by a distance of at most $2h_{\text{opt}}$. Thus, any given point in the pattern can be moved by at most h_{opt} during the translation and at most $2h_{\text{opt}}$ during the rotation, and could have been initially at most h_{opt} away from its associated background point. Therefore, each point in the pattern will be at most a distance of $4h_{\text{opt}}$ from a background point. The pattern in its current position coincides with one of the Euclidean transformation generated and checked by our algorithm. ∎

2.3 Rotation in $\Re^3$

Suppose we are given a set B of n points in $\Re^3$ and a set P of m points in $\Re^3$. In this subsection we give an efficient method for finding a (pure) rotation R', about the origin, such that $h(R'(P), B)$ is at most $4h(R_{\text{opt}}(P), B)$, where $R_{\text{opt}}(P)$ is an *optimal* rotation, i.e., one that minimizes $h(R(P), B)$, taken over all rotations R.

Find a point $p_1 \in P$ that is furthest from the origin. Find a point $p_2 \in P$ that maximizes its perpendicular distance to the line defined by the origin and point p_1. For each $b' \in B$, define $R1_{b'}$ to be the rotation that makes the origin, p_1 and b' collinear. For each $b'' \in B, b'' \neq b'$, define $R2_{b''}$ to be the rotation about the origin-p_1 axis that makes the origin, p_1, p_2, and b'' coplanar. Our method, then, is to find $\min_{b',b'' \in B}\{h(R2_{b''}(R1_{b'}(P)), B)\}$ as our best approximate match, and let R' be the resultant rotation $R2_{b''}(R1_{b'}(P))$ that achieves this bound. This can clearly be done in $O(n^2m * Nearest_3(n))$ time, which is $O(n^2 m \log^2 n)$ if one uses the best current point location method for a 3-dimensional convex subdivision [15] to query nearest neighbors in a 3-dimensional Voronoi diagram (e.g., see [12, 23, 24]). Our preference, however, is to achieve a faster (and more practical) $O(n^2 m \log n)$ time bound using the approximate nearest neighbors method of Arya *et al.* [5]. This slightly weakens our approximation factor, as we show in the following lemma, but we feel this is justified by the time efficiency.

Lemma 2.3 *$h(R'(P), B) \leq (4 + \epsilon)h(R_{\text{opt}}(P), B)$, for any constant $0 < \epsilon < 1$.*

Proof: Since the ϵ term is a direct consequence of our using approximate nearest neighbor searching to achieve $Nearest_3(n) = O(\log n)$, it is sufficient to show that actual nearest neighbors would give an approximation factor of 4. For simplicity of expression, define $h_{\text{opt}} = h(R_{\text{opt}}(P), B)$. Observe that for each $p \in R_{\text{opt}}(P)$, there exists an associated $b \in B$ that is within a distance h_{opt} of p. Consider the process of rotating the entire pattern $R_{\text{opt}}(P)$ so that p_1, the furthest pattern point from the origin, now becomes collinear with its associated background point b and the origin. This process can move any point in the pattern by at most h_{opt}. Now consider the second rotation that aligns p_2 with its matching background point. This rotation may move p_2 a distance of at most $2h_{\text{opt}}$, and therefore it may move any point in the pattern by at most $2h_{\text{opt}}$. This rotation will be one of those generated and checked by our algorithm. ■

2.4 Translation and Rotation in $\Re^3$

Suppose we are given a set B of n points in $\Re^3$ and a set P of m points in $\Re^3$. In this subsection we give an efficient method for finding a Euclidean transformation E' such that $h(E'(P), B)$ is at most $(8 + \epsilon)h(E_{\text{opt}}(P), B)$, where $E_{\text{opt}}(P)$ is the *optimal* Euclidean transformation, i.e., the one that minimizes $h(E(P), B)$, taken over all such transformations E.

Select from the pattern diametrically opposing points and call them r and k. Choose a point $l \in P$ such that the perpendicular distance from l to the line rk is maximum. For each $b \in B$, define T_b to be the translation that takes pattern point r to b.

For each $b' \in B, b' \neq b$, define $R1_{b'}$ to be the rotation that causes r, k and b' to become collinear. For each $b'' \in B, b'' \neq b', b'' \neq b$, define $R2_{b''}$ to be the rotation about the rk axis that brings b'' into the r, k, l plane.

Our method, then, is to compute the value of $\min_{b,b',b'' \in B}\{h(R2_{b''}(R1_{b'}(T_b(P))), B)\}$ as our best approximate match, and let E' be the Euclidean transformation $R2_{b''}(R1_{b'}(T_b(P)))$ that achieves this bound. This can be done in $O(n^3 m * Nearest_3(n))$ time. As above, we achieve $Nearest_3(n) = O(\log n)$ using approximate nearest-neighbor searching [5], and end up with the following result:

Lemma 2.4 *$h(E'(P), B) \leq (8 + \epsilon)h(E_{\text{opt}}(P), B)$, for any constant $0 < \epsilon < 1$.*

Proof: For simplicity of expression, define $h_{\text{opt}} = h(E_{\text{opt}}(P), B)$. In addition, as in previous proofs, let us show that the expansion factor is 8 if one were to use actual nearest neighbors instead of approximate nearest neighbors. Observe that for each $p \in E_{\text{opt}}(P)$, there exists an associated $b \in B$ that is within a distance h_{opt} of p. Consider the process of translating the entire pattern $E_{\text{opt}}(P)$ so that r becomes coincident with its associated background point. This process can move any point in the pattern by a distance of at most h_{opt}. Now consider the process of rotating the entire pattern so that line rk passes through the background point that was associated with k. This rotation can move any point in the pattern by at most $2h_{\text{opt}}$. Now consider a second rotation that brings the background point associated with l into the rlk plane. This rotation may move pattern point p_2 a distance of at most $4h_{\text{opt}}$, and therefore it may move any point in the pattern by at most $4h_{\text{opt}}$. This rotation will be one of those generated and checked by our algorithm. The translation may have moved any point a distance of at most h_{opt}, the first rotation may have moved any point a distance of at most $2h_{\text{opt}}$ farther, and the second rotation may have moved any point a distance of at most $4h_{\text{opt}}$ still farther. Considering that any given pattern point may have been a distance of h_{opt} away from its associated background point to start with, no pattern point can be farther than $8h_{\text{opt}}$ from its associated background point. Our algorithm will therefore produce a Euclidean transformation that has a directed Hausdorff distance that is at most a factor of eight times the minimal. ■

Thus, we have shown how one can easily achieve all the results claimed in Table 1.

3 The Experimental Setup

We have claimed more than this, however, in that we assert that our methods are *practical.* Therefore, so as to justify this stronger claim, we have produced an implementation of our methods, and we have compared this implementation with a method that produces best matches to an arbitrary precision using a conventional branch-and-bound search of a discretized configuration space. This conventional method seems to be the most practical previous best match procedure (we did not feel it was practically feasible to implement the previous intersection-based methods). As we show through our experimental results, however, this conventional method is still quite slow compared to our method, and the matches it finds are not that much better than the ones that our method finds.

3.1 Example Generation

We generated problem instance examples in dimension d by randomly choosing a set of n background points in the unit *d-cube* with lower left corner at the origin. The points are generated by selecting each of the d coordinates from the range $[0, 1)$ with uniform distribution. Once we generate this background, we randomly select m points from this background to be an unperturbed pattern. Each pattern point is then perturbed by a small amount so that the pattern no longer identically resembles a subset of the background points. The perturbation is done as follows. A parameter δ defines the maximum distance that a pattern point can be moved. The original position of a pattern point and the distance δ define a hypersphere centered at the point's original position. The pattern point will be moved from its original position to a point inside of this *d-sphere* such that the probability that it will be placed in any subvolume of the sphere is equal to the ratio of that subvolume to the volume of the entire *d-sphere*. In other words, perturbation causes each pattern point, p to be randomly placed with uniform distribution in the hypersphere with center p and radius δ. After this perturbation process, the pattern is referred to as a *perturbed pattern*. We use this perturbed pattern and the background from which it was generated as the sets P and B, respectively.

3.2 Implementation of the Approximate Match Algorithms

This section gives the details of the implementation of our approximate-match algorithm for four different matching problems involving translation and rotation in 2 and 3 dimensions.

3.2.1 Translation in $\Re^2$

As described in Section 2.1, our approximate pattern matching algorithm translates the pattern so that the distinct representative of the pattern coincides with each of the n background points in succession. For each such translation, the Hausdorff distance is calculated and compared with the best found so far. If the new Hausdorff distance is smaller that the best found so far, the position of the pattern (i.e., the position of the distinct representative) and this new best distance replace those recorded so far. After the pattern has been translated to all of the background points, we output the best found. This position of the pattern is known to produce a Hausdorff distance that is within a factor of two of optimal.

There are several heuristics one can apply to this method, which do not improve the worst-case running time, but which do improve the running time in practice. First, since the relative qualities of the various translations are identical using ρ to be either the distance between the points or the distance squared between the points, we use distance squared to avoid the costly square root function in this inner-most loop. At the end of the program, when we have found the translation that we want, we then take the square root to get the actual Hausdorff distance.

We also use a condition that terminates the while-loop early once it is known that a particular placement need not be further considered. Observe that in the calculation of the Hausdorff distance, we are finding the maximum amount by which a pattern point deviates from its nearest background point. As we determine this quantity for each of the pattern points we have a partial-maximum at any given point in the loop. If this partial-maximum ever exceeds the best Hausdorff distance found so far, the placement that we are checking is known to be suboptimal and does not warrant fruther consideration. We therefore terminate the while loop as soon the partial computation of the Hausdorff distance exceeds the global best found so far.

3.2.2 Translation in $\Re^3$

The algorithm for translation in $\Re^3$, and for that matter $\Re^d$, is identical to the algorithm for translation in $\Re^2$, except that the distances are computed in a higher dimension.

3.2.3 Translation and Rotation in $\Re^2$

The implementation of the algorithm described in Section 2.2, is also quite straightforward. Diametrically opposing pattern points are chosen, one of which will serve as both the distinct representative of the pattern and as also the center of rotation. The algorithm then translates the pattern so that the distinct representative coincides with each of the n background points in succession. After each translation, the pattern is rotated about the current position of the distinct representative a total of $n - 1$ times so that after each rotation, the other antipodal point is aligned with another one of the background points. We now have the pattern in one of the $n * (n - 1)$ positions at which we check the Hausdorff distance. As with the translation-only case, we maintain the best Hausdorff distance found so far and the position of the pattern that produced it. If at any time one of the $n * (n - 1)$ placements has a directed Hausdorff distance that is better than the best found so far, our records are updated to reflect this new best position and Hausdorff distance. Again, we use the distance-squared and early loop-termination heuristics in order to further speed up the algorithm.

3.2.4 Translation and Rotation in $\Re^3$

The algorithm for producing an approximate match for translation and rotation in $\Re^3$ is as follows. We again select the distinct representative and the antipode of the pattern as we have done in Section 3.2.3. In this case, we also select a third pattern point, called the radial point, which has the property that it is the greatest distance away from the line passing through the distinct representative and the antipode. Our approximate match algorithm is comprised of three nested for-loops. The outer-most loop translates the pattern such that the distinct representative of the pattern coincides with each of the n background points in succession. The next loop chooses one of the remaining $n-1$ background points and rotates the pattern about the current position of the distinct representative so that the antipode becomes aligned with this selected background point. The inner-most for-loop selects a third background point from the remaining $n-2$ and performs a second rotation of the pattern, this time about the line passing through the current position of the distinct representative and the current position of the antipode, to bring the plane defined by the distinct representative, the antipode, and the radial point into a position that includes the background point chosen by this third for-loop. Note that there are two rotations summing to 2π that that will do this, one bringing the radial point nearer to the third chosen background point and the other taking it farther away. We are of course interested in the one that brings it closer, which is the rotation that has the smaller absolute value.

For each of the $n(n-1)(n-2)$ placements produced by the above described for loops, the Hausdorff distance of the placement is generated and the current best is kept. At the termination of our algorithm, we output the best placement found.

Implementation of the rotations described in this section are explained in detail in Foley and vanDam [13].

3.3 Implementation of the Branch and Bound Algorithms

This section gives the implementation details of the corresponding branch-and-bound algorithm for matching problems involving translation and rotation in 2 and 3 dimensions.

3.3.1 The Conventional Method for Translation in $\Re^2$

The conventional method against which we compared our method is a recursive algorithm. It receives a square defined by a center point and a side length. It then "probes" the center of the square by translating the pattern so that the distinct representative of the pattern is in the center of the square. For the pattern in this position, the Hausdorff distance is calculated. If this distance is the best found so far, it is recorded along with the probe point (center of square). The algorithm then recurses on each of its four quadrants.

Notice that squares that are passed to the recursive algorithm will in general exclude both pattern points and background points. The pattern and background are not cropped to fit these squares, these squares simply describe areas in which we would consider placing the distinct representative of the pattern in an effort to find good matches. The recursion is terminated when it reaches a predefined maximum depth or if it is certain that placement of the distinct representative at any point in the square will not produce a Hausdorff distance that is better than the best found so far. One observation that we can use to terminate the depth of recursion early is that the Hausdorff distance can be decreased by an amount of at most x when the pattern is translated by a distance of x. If the value of the Hausdorff distance produced by probing the center of the square is so great relative to the best found so far that placing the distinct representative at any point in the square is known to produce a Hausdorff distance that does not beat the best found so far, we no longer need to recursively search this square and we can terminate this branch of the recursion.

In the approximate match algorithm, we allow the condition an early exit to the while loop if the Hausdorff distance of a particular placement is known to be suboptimal. In the conventional algorithm, we do not. The Hausdorff distances are used as probe values and large probe values have a tendency to terminate recursion early. We do not use this break out of the while loop since it would return to the recursive algorithm probe values which were, in most cases, smaller than those that are accurate. Such smaller probe values inhibit pruning and we find that is better to allow the probing algorithm to run to completion and return accurate probe values.

3.3.2 The Conventional Algorithm for Translation and Rotation in $\Re^2$

The conventional method for Translation and Rotation in $\Re^2$ involves searching the three-dimensional configuration space in which the x and y position of the distinct representative of the pattern comprise two of the dimensions and the angular position of the pattern about the distinct representative, θ, comprises the third.

We search the configuration space defined by $x, y, \in [0,1]$ and the angle $\theta \in [-\pi, \pi]$. We probe the center of this volume by translating the pattern so that the distinct representative resides at point $(0.5, 0.5)$ and rotate the pattern about the distinct representative so that the antipode of the pattern is at 0 radians in relation to the distinct representative. We then calculate the directed Hausdorff distance of the pattern in this position and if it is better than the best found so far, we record it along with the probe point, x, y, and θ that produced it. We then recurse on each of the 8 octants until the maximum depth of recursion is reached or it is certain that no placements in the configuration space that we are currently searching will have a directed Hausdorff distance that is better than the global best found so far, in which case this branch of the recursion is terminated.

3.3.3 Translation and Rotation in $\Re^3$

The conventional method for Translation and Rotation in $\Re^3$ is again the search of a configuration space, this time a 6-dimensional space. One has three degrees of freedom in placing the distinct representative of the pattern relative to the background, specifically the x, y, and z position, and one has three degrees of freedom in orienting the pattern in three-space once the distinct representative has been pinned. Two of these degrees of freedom involve the location of the antipode of the pattern relative to the distinct representative, which can be thought of as the ϕ and θ components of the spherical coordinate of the antipode in relation to the distinct representative. The final degree of freedom is the angular orientation of the pattern about the axis from the distinct representative and the antipode.

This conventional algorithm is similar in nature to the two previously presented. The center of the 6-space is probed, by placing and orienting the pattern according to the value for each of the 6 variables defining a placement that are contained in the probe point. The directed Hausdorff distance of the pattern in this position is calculated and the algorithm recurses on each of the $2^6 = 64$ subregions of the configuration space until either the maximum depth of recursion has been reached or until pruning occurs.

3.4 Experiment 1: Ratio of approximate to actual for Translation in $\Re^2$

For the case of translation, we have provided an upper bound on the directed Hausdorff distance of the approximate match algorithm of twice that of the directed Hausdorff distance of the *optimal* match. It is likely, in practice, however, that the approximate match algorithms will produce matches with better directed Hausdorff distances than this worst case. It is our conjecture that for large sparse B's and large sparse P's, the approximate match algorithms will produce matches that are $(1 + \lambda) * h_{\text{opt}}$, where λ is the ratio of the expected distance by which a point will be perturbed divided by the maximum distance by which a point will be perturbed. In particular, for our perturbation strategy in two dimensions, λ should be close to $1/\sqrt{2}$, since this is the expected distance the representative point p is from its "match point" in B, and it is likely that there is another point in P that is close to h_{opt} from its match point in a way that it gets moved the maximum amount possible when we move p to its match point.

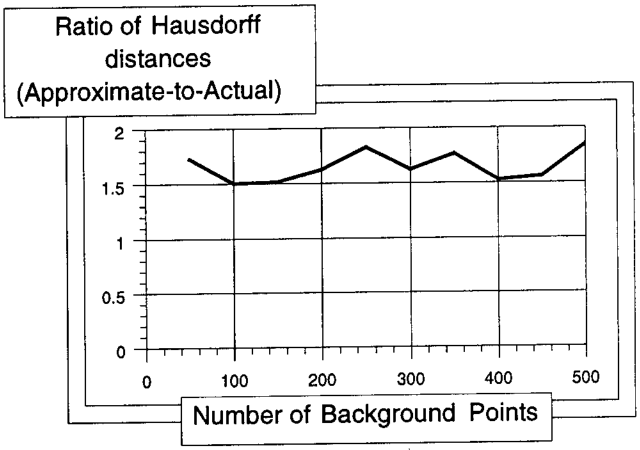


Figure 1: Ratios of approximation method to conventional method.

We conducted an experiment to test this. Thirty sets of background points were generated, each having between 50 and 500 points. From each background, a pattern of size 10 was selected and perturbed. The pattern was then matched to its associated background using both the approximate match algorithm and a conventional match algorithm. The ratio of the directed Hausdorff distance of the approximate match algorithm to the conventional match algorithm is plotted in Figure 1. The average of the ratios plotted is 1.69, which is close to the predicted value of 1.71.

3.5 Experiment 2: Ratio of approximate to actual for Translation in $\Re^3$

In a manner similar to Experiment 1, it could be conjectured that the matches produced by the approximate match algorithm in three dimensions will be close to $1/\sqrt[3]{2}$, since this is the average distance by which a point is perturbed with our perturbation strategy relative the maximum distance with which a point will be perturbed with our perturbation strategy.

Experiment 3 was conducted to test this. Forty-five sets of background points were generated, each having between 5 and 100 points. From each background, a pattern of size 5 was selected and perturbed. The pattern was then matched to its associated background using both the approximate match algorithm and a conventional match algorithm. The ratio of the directed Hausdorff distance of the approximate match algorithm to the conventional match algorithm is plotted in Figure 2. The average of the ratios plotted is 1.679, which is close to the predicted value of 1.79. The fact that the experimental average is less than the predicted value can be attributed to the relatively small pattern size of

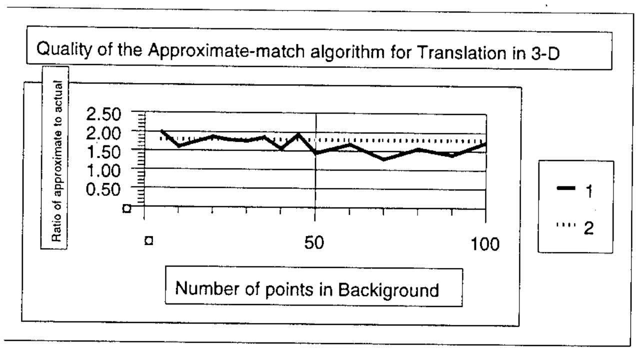


Figure 2: Ratios of approximation method to conventional method Translation only in 3D.

5 points, which decreases the probability that one of the pattern points will be translated by a nearly maximal amount when the distinct representative of the pattern is translated to its associated background point.

3.6 Experiment 3: Running Times

The approximate match algorithms have worst-case time complexities that are much lower than those of the actual match algorithms. We conjectured, however, that they should run faster in practice. This experiment examines the extent to which they do, comparing them to what should be a good practical algorithm—branch-and-bound. This experiment involves two cases, the translation-only case in $\Re^2$ and the translation-and-rotation case in $\Re^3$. So as to eliminate any system effects on the running time data, we keep a counter of floating point operations used by each algorithm as opposed to using the Unix *time* command. The numbers presented in the graphs are numbers obtained from this counter.

In the translation-only case in $\Re^2$, thirty sets of background points were generated, each having between 50 and 500 points. From each background, a pattern of size 10 was selected and perturbed. The pattern was then matched to its associated background using both the approximate match algorithm and a conventional match algorithm both with and without heuristic speedups. The results of this experiment are shown in Figure 3. In every case the approximate match algorithm, even without the heuristic speedup, had a smaller running time than the conventional algorithms.

In the translation-and-rotation case in $\Re^3$, 24 sets of background points were generated, each having between 5 and 25 points. From each background, a pattern of size 5 was selected and perturbed. The pattern was then matched to its associated background using the approximate match algorithm with heuristic speedups, a depth-first branch-and-bound algorithm and a breadth-first branch-and-bound algorithm.

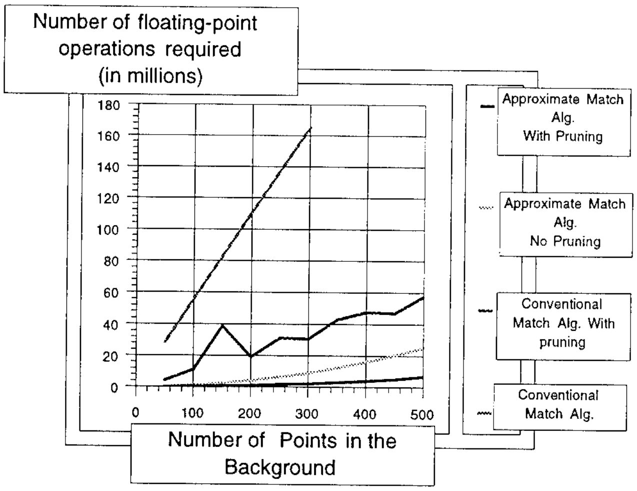


Figure 3: Running times of the four methods.

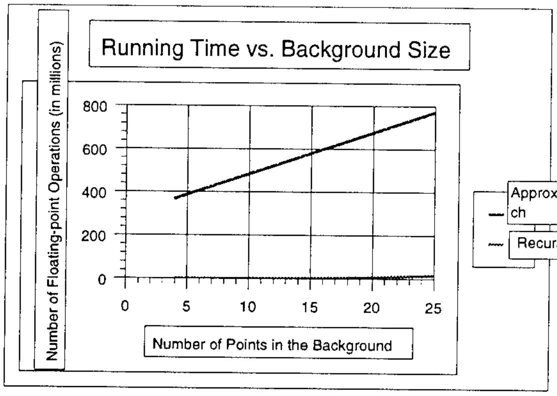


Figure 4: Running times of the two 3-D methods. Line 1 graphs the running times for the conventional method and line 2 graphs the running times for our method

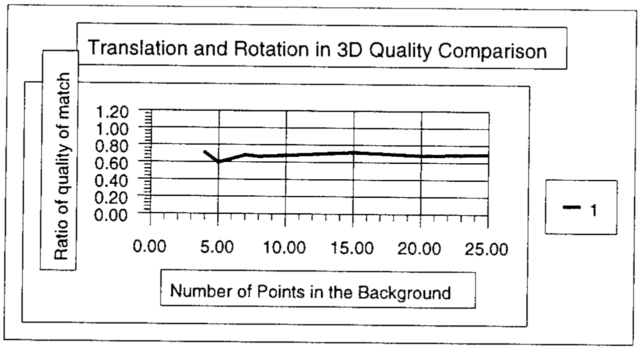


Figure 5: Running times of the four methods.

The running times of the depth-first and breadth-first branch-and-bound algorithms were, in all instances within a factor of 0.01 of each other and are therefore plotted as a single line in Figure 4 which depicts the results of this experiment.

The branch-and-bound algorithms search a six dimensional space comprised of three degrees of freedom in the translation of the pattern and three degrees of freedom in the rotation of the pattern. This produces a rather large branching factor $2^6 = 64$ in the recursive algorithms, and necessitated the depth of these algorithms to be limited to 3. With this (necessary) depth limitation, the approximate-match algorithm actually found better matches than the branch-and-bound algorithm did in all of the 24 cases, in spite of the fact that the branch-and-bound algorithms required on average 4479 times as many floating-point operations. The average directed Hausdorff distance of the match producted by the approximate-match algorithm and the average directed Hausdorff distance of the match produced by the branch-and-bound algorithms are given in Figure 5 for comparison.

3.7 Experiment 3: Running Times vs. Depth of Recursion

The depth of search of the conventional match algorithms that we have implemented must be limited. This experiment depicts the extent to which the running time of the algorithm increases as the depth of recursion is increased. Further, it shows the substantial speedup obtained by pruning the search. The results of this experiment are depicted in Figure 6.

Four backgrounds were generated, each having 50 points. From each background, a pattern of size 10 was selected and perturbed. The conventional method without pruning was run ten times on the first background, with the depth of recursion being varied from 1 to 10. The conventional method with pruning was run ten times on the next three backgrounds, and again ten times on each, with the recursion depth being varied from 1 to 10. An average of the running times for the three cases with pruning were taken and the results were plotted.

4 Conclusion

We have given approximate pattern matching algorithms for translation, rotation, and Euclidean transformations for two, three, and sometimes even higher dimensions. These matching algorithms are guaranteed to give a match with a directed Hausdorff distance that is no greater than a small constant times the best achievable directed Hausdorff distance. In addition, they have a time complexity that is substantially smaller that those of existing pattern matching algorithms, they are easy to implement, and they run fast in practice, as predicted.

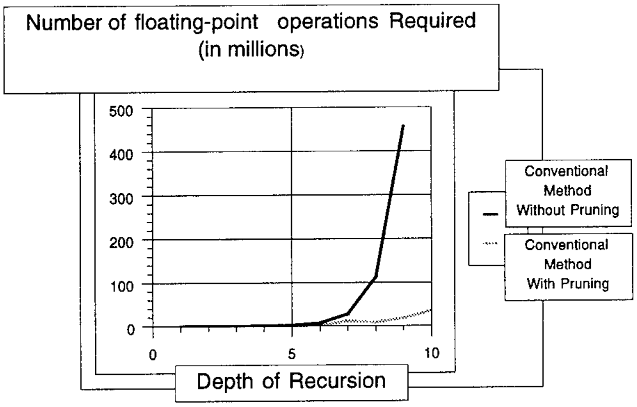


Figure 6: Running times with and without pruning.

Acknowledgements

We thank Esther Arkin, Sándor Fekete, and David Mount for useful discussions on this research. Some of the results grew out of discussions during a Workshop on Geometric Probing in Computer Vision, sponsored by the Center for Night Vision and Electro-Optics, Fort Belvoir, Virginia, and monitored by the U.S. Army Research Office. The views, opinions, and/or findings contained in this report are those of the authors and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

References

[1] P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri, "Selecting distances in the plane," in *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, 321–331, 1990.

[2] H. Alt, B. Behrends, and J. Blömer, "Approximate matching of polygonal shapes," in *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, 186–193, 1991.

[3] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl, "Congruence, similarity and symmetries of geometric objects," *Discrete Comput. Geom.*, **3**, 237–256, 1988.

[4] E. M. Arkin, K. Kedem, J. S. B. Mitchell, J. Sprinzak, and M. Werman, "Matching points into pairwise-disjoint noise regions: combinatorial bounds and algorithms," *ORSA J. Comput.*, 4(4), 375–386, 1992.

[5] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu, "An optimal algorithm for approximate nearest neighbor searching," in *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, 1994.

[6] L. P. Chew, M. Goodrich, D. Huttenlocher, K. Kedem, J. Kleinberg, and D. Kravets, "Geometric pattern matching under Euclidean motion," in *Proc. 5th Canad. Conf. Comput. Geom.*, Waterloo, Canada, 151–156, 1993.

[7] L. P. Chew and K. Kedem, "Improvements on geometric pattern matching problems," in *Proc. 3rd Scand. Workshop Algorithm Theory, Lecture Notes in Computer Science*, vol. 621, Springer-Verlag, 318–325, 1992.

[8] R. Cole, "Slowing down sorting networks to obtain faster sorting algorithms," in *Proc. 25th Annu. IEEE Sympos. Found. Comput. Sci.*, 255–260, 1984.

[9] R. Cole, "Parallel merge sort," *SIAM J. Comput.*, **17**(4), 770–785, 1988.

[10] R. Cole, J. Salowe, W. Steiger, and E. Szemerédi, "An optimal-time algorithm for slope selection," *SIAM J. Comput.*, **18**, 792–810, 1989.

[11] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Mass., 1990.

[12] H. Edelsbrunner, *Algorithms in Combinatorial Geometry, EATCS Monographs on Theoretical Computer Science*, vol. 10, Springer-Verlag, Heidelberg, West Germany, 1987.

[13] J. D. Foley and A. Van Dam, *Fundamentals of Interactive Computer Graphics*, Addison-Wesley, Reading, MA, 1982.

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, NY, 1979.

[15] M. Goodrich and R. Tamassia, "Dynamic trees and dynamic point location," in *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, 523–533, 1991.

[16] P. J. Heffernan, "Generalized approximate algorithms for point set congruence," in *Proc. 3rd Workshop Algorithms Data Struct., Lecture Notes in Computer Science*, vol. 709, 373–384, 1993.

[17] P. J. Heffernan and S. Schirra, "Approximate decision algorithms for point set congruence," in *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, 93–101, 1992.

[18] D. P. Huttenlocher and K. Kedem, "Computing the minimum Hausdorff distance for point sets under translation," in *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, 340–349, 1990.

[19] D. P. Huttenlocher, K. Kedem, and J. M. Kleinberg, "On dynamic Voronoi diagrams and the minimum Hausdorff distance for point sets under Euclidean motion in the plane," in *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, 110–120, 1992.

[20] D. P. Huttenlocher, K. Kedem, and M. Sharir, "The upper envelope of Voronoi surfaces and its applications," in *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, 194–203, 1991.

[21] K. Imai, S. Sumino, and H. Imai, "Minimax geometric fitting of two corresponding sets of points," in *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, 266–275, 1989.

[22] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms," *J. ACM*, **30**, 852–865, 1983.

[23] K. Mehlhorn, *Multi-dimensional Searching and Computational Geometry, Data Structures and Algorithms*, vol. 3, Springer-Verlag, Heidelberg, West Germany, 1984.

[24] F. P. Preparata and M. I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, New York, NY, 1985.

[25] G. Rote, "Computing the minimum Hausdorff distance between two point sets on a line under translation," *Inform. Process. Lett.*, **38**, 123–127, 1991.

[26] W. Rucklidge, "Lower bounds for the complexity of the Hausdorff distance," in *Proc. 5th Canad. Conf. Comput. Geom.*, Waterloo, Canada, 145–150, 1993.

Spheres, Molecules, and Hidden Surface Removal*

Dan Halperin[†] Mark H. Overmars[‡]

Abstract

We devise techniques to manipulate a collection of loosely interpenetrating spheres in three-dimensional space. Our study is motivated by the representation and manipulation of molecular configurations, modeled by a collection of spheres. We analyze the sphere model and point to its favorable properties that make it more easy to manipulate than an arbitrary collection of spheres. For this special sphere model we present efficient algorithms for computing its union boundary and for hidden surface removal. The efficiency and practicality of our approach are demonstrated by experiments on actual protein data.

1 Introduction

In this paper we devise techniques to represent and manipulate a collection of overlapping spheres in three-dimensional space. Often, manipulating three-dimensional geometric objects that intersect is time consuming. However, by imposing several constraints on the spheres and their interaction, we obtain a setting where efficient and practical techniques are quite easy to obtain. Our study is motivated by the representation and manipulation of molecular configurations, modeled by a collection of spheres.

A common approach to representing the three-dimensional geometric structure of a molecule, is to represent each of its atoms by a "hard" sphere. It is also common to assume that the *nuclear arrangement*, i.e., the relative displacement of the spheres, is fixed (it is often the so-called equilibrium nuclear configuration). There are recommended values for the radius of each atom sphere and for the distance between the centers of every pair of spheres. In this model, the spheres are allowed to interpenetrate one another, therefore it is sometimes referred to as the "fused spheres" model (see, e.g., Figure 1). The envelope surface of the fused spheres may be regarded as a formal *molecular surface.* It is evident, that various properties of molecules are disregarded in this simple model. However, in spite of its approximate nature, it has proven useful in many practical applications. For more background material and references, see, e.g., the survey paper by Mezey [16].

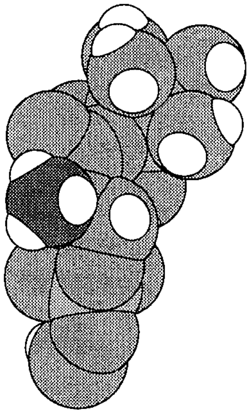


Figure 1: Molecule 1: A small molecule surrounded by water with a total of 42 atoms.

*The first author acknowledges support by a Rothschild Postdoctoral Fellowship, by a grant from the Stanford Integrated Manufacturing Association (SIMA), and by NSF/ARPA Research Grant IRI-9306544. The second author was partially supported by the ESPRIT III Basic Research Action No. 7141 (project ALCOM II) and by the Netherlands Organization for Scientific Research (N.W.O.).

[†]Robotics Laboratory, Department of Computer Science, Stanford University, Stanford, CA 94305, USA. Email: halperin@cs.stanford.edu.

[‡]Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB, Utrecht, the Netherlands. Email: markov@cs.ruu.nl.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

We study the hard sphere model from a *computational geometry* point of view, that is, we study the combinatorial and algorithmic behavior of a collection of n (possible intersecting) spheres in 3-space, having some special properties. We make several simple observation, showing that, because of these special properties, the spheres in this model can be efficiently manipulated. For example, we show that the maximum combinatorial complexity of the boundary of the union of the fused spheres is $O(n)$, whereas for an arbitrary collection of spheres (even unit spheres) this may be $\Theta(n^2)$ (see, e.g., [13]). These results are described in Section 2.

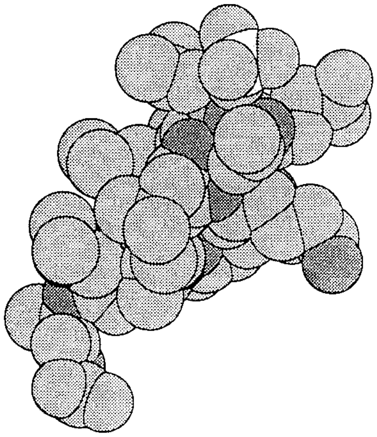


Figure 2: Molecule 2: A small protein with 100 atoms.

In Section 3 we take advantage of the favorable behavior of the model, to devise a data structure for answering *intersection queries* of the form: Given a hard sphere model M of a molecule and a query atom sphere Q (both given in a fixed placement in 3-space), report which spheres of M are intersected by Q. In Section 4 we use this data structure to efficiently compute the boundary of the union of the spheres. We also show that this algorithm can be used to compute a related type of molecular surfaces, so-called *solvent accessible surfaces.* Finally, in Section 5, we present efficient algorithms for hidden surface removal, either by computing a depth order or by computing the visibility map of a molecule.

Our work is closely related to the study of *fatness.* It has been shown that certain problems in computational geometry can be solved much more efficiently when the objects involved have no long and thin parts (see, e.g., [15, 17, 20]). Clearly, spheres are fat. In 3-space though fatness is not enough to guarantee efficient algorithms when the objects are allowed to intersect. Fortunately, the extra properties in the hard sphere model provide enough additional constraints

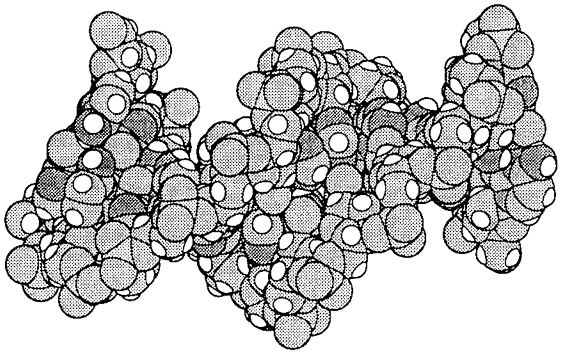


Figure 3: Molecule 3: A piece of DNA with 757 atoms.

on the objects to allow for efficient algorithms. It exemplifies, yet another time, that in practical situations a collection of geometric objects having certain additional properties (obtained from observing real-life situations), behaves favorably.

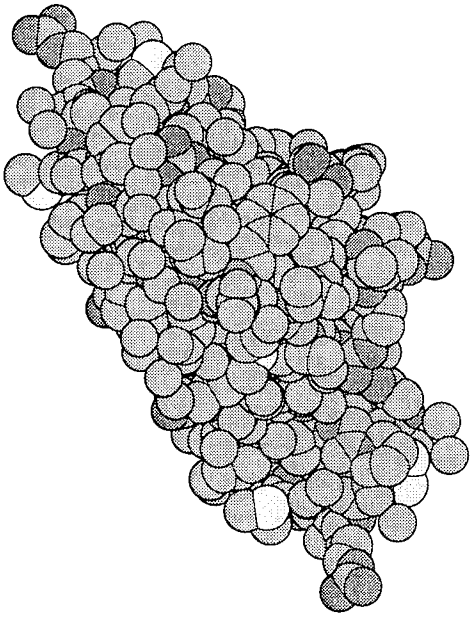


Figure 4: Molecule 4: A larger protein with 971 atoms.

We support our claim for efficiency and practicality of the data structure and the algorithms that we present by reporting results of experiments that were carried out on actual protein data retrieved from the "Protein Data Bank" at Brookhaven National Laboratory [1, 2]. In Figures 1 till 4 you find pictures of some of the molecules that we used. All of these pictures were created using the algorithms described in Section 5.

2 The Hard Sphere Model

Overlapping spheres in three-dimensional space may in general be rather unwieldy objects. The *arrangement* (i.e., the subdivision of space) defined by n spheres in 3-space may have complexity $\Theta(n^3)$ in the worst case and their union boundary may have complexity $\Theta(n^2)$. Moreover, efficient algorithms for manipulating a collection of overlapping spheres are often complex and rather time consuming. However, the atom spheres in the model of a molecule have properties that we can exploit to obtain efficient and simple algorithms for manipulating them. One such property is that, although two spheres may interpenetrate, their centers cannot get too close to one another. The other useful property is that their radii range in a fairly restricted range; see, e.g., [11, p. J-3] for a list of van der Waals radii, which are one type of acceptable radii. Actually, each molecular modelling package seems to use its own slightly different set of radii. The following table shows the list the radii (in Ångstrøm) of the spheres we use to represent the main types of atoms. (Taken from an actual molecular modelling package.)

C	Cal	H	N	O	P	S
1.70	2.07	0.90	1.55	1.52	1.80	1.80

In Theorem 2.1 below, we state the conditions that make the sphere model of a molecule favorable. (Similar observations were recently, independently made by Varshney and Brooks [21].) From this point on, we will distinguish between a *ball* and a *sphere*. Each atom with radius r_i and center at c_i induces a ball $B_i = \{p | d(p, c_i) \le r_i\}$ and a sphere $S_i = \{p | d(p, c_i) = r_i\}$, where $d(p_1, p_2)$ is the Euclidean distance between the points p_1 and p_2 in 3-space.

Theorem 2.1 *Let $M = \{B_1, \ldots, B_n\}$ be a collection of n balls in 3-space with radii $r_1, \ldots, r_n$ and centers at $c_1, \ldots, c_n$. Let $r_{\min} = \min_i r_i$ and let $r_{\max} = \max_i r_i$. Also let $S = \{S_1, \ldots, S_n\}$ be the collection of spheres such that S_i is the boundary surface of B_i. If there are positive constants k, ρ such that $\frac{r_{\max}}{r_{\min}} < k$ and for each B_i the ball with radius $\rho \cdot r_i$ and concentric with B_i does not contain the center of any other ball in M (besides c_i), then:*

(i) *For each $B_i \in M$, the maximum number of balls in M that intersect it is bounded by a constant.*

(ii) *The maximum combinatorial complexity of the boundary of the union of the balls in M is $O(n)$.*

Proof. Part (i): Consider a ball B_i. Let B be the ball of radius $r_i + 2r_{\max}$, centered at c_i. Clearly, any ball in M that intersects B_i must lie completely inside B. For each B_j that lies completely inside B, let β_j denote the ball of radius $\frac{\rho}{2} \cdot r_{\min}$ centered at c_j. We claim that the β_j's are pairwise interior-disjoint. Suppose the opposite, namely, that there are two such balls β_{j_1} and β_{j_2} whose interiors intersect. This implies that c_{j_1} and c_{j_2} are less than $\rho \cdot r_{\min}$ apart. This, in turn, means that the ball with radius $\rho \cdot r_{j_1}$ and center at c_{j_1} contains the center of the ball B_{j_2}, contradicting the assumptions of the theorem.

Hence, by volume consideration, the total number of balls that are completely contained in B cannot exceed

$$\frac{(r_i + 2r_{\max})^3}{(\frac{\rho}{2} \cdot r_{\min})^3} \le \frac{(3r_{\max})^3}{(\frac{\rho}{2} \cdot r_{\min})^3} \le 216 \cdot (\frac{k}{\rho})^3.$$

Part (ii) follows from Part (i) because Part (i) implies that the number of features involving B_i on the union boundary is bounded by a constant. Indeed, if the balls are in general position then a feature involving a ball B_i is either a face on the boundary of B_i, or an intersection arc of B_i with another ball, or an intersection point of B_i with two other balls. All these features belong to the two-dimensional subdivision (or *arrangement*) formed on the boundary of B_i by a constant number of circles, each being the intersection of B_i with another ball. The complexity of this arrangement is evidently bounded by a constant. Thus, the overall complexity of the union boundary is $O(n)$. □

The following table gives the values of k, ρ and the maximal and average number of balls intersecting a single ball for our four example molecules (Figures 1 to 4).

mol.	k	ρ	max	aver.
1	2.30	0.64	14	5.0
2	1.18	0.71	12	6.4
3	2.00	0.63	15	7.4
4	1.18	0.71	12	6.3

As can be seen k is small and ρ is large as required, resulting in a small number of intersections per ball (being much smaller than the worst-case bound of $216(k/\rho)^3$).

We now turn to discuss the subdivision of three-dimensional space induced by a collection of spheres; the *arrangement* of the spheres. The study of arrangements plays a fundamental role in geometric computing. We denote the arrangement of the spheres in M by $\mathcal{A}(M)$, that is, $\mathcal{A}(M)$ is the subdivision of 3-space into cells of dimensions $0, 1, 2$ and 3, induced by the spheres in M (see, e.g., [4, pp. 144–150]). The combinatorial complexity of the arrangement $\mathcal{A}(M)$ is defined to be the number of cells of various dimensions in this subdivision. For arbitrary sets of spheres the complexity can be $\Theta(n^3)$. The following is an immediate corollary of Theorem 2.1 (i):

Corollary 2.2 *The complexity of the arrangement $\mathcal{A}(M)$ of the spheres in M as defined in Theorem 2.1 is $\Theta(n)$.*

For most algorithmic uses, however, a raw arrangement is an unwieldy structure. The difficulty is that many cells in an arrangement can have very complex topologies, and thus navigating around them is difficult. What we often want is a further refinement of the cells of an arrangement into pieces that are each homeomorphic to a ball and have constant description complexity.

A prevailing and general technique for decomposing arrangements is the *vertical decomposition* (see, e.g., [3]), defined as follows. For every sphere S_i, we call the curve of intersection of S_i and the horizontal plane through the center of S_i the *equator* of the sphere. Let E be the collection of curves on the spheres in M consisting of intersection curves between any pair of spheres and the equators of the spheres. Let γ be a curve in E. We extend a vertical segment upward and downwards from every point of γ until it hits a sphere in M or extends to infinity. We repeat this process for every curve in E. As a result we get a collection of vertical walls that together with the spheres in M subdivide 3-space into xy-monotone 3D cells. We then project each cell onto the xy-plane, and extend vertical segments (with respect to the y-direction) from every vertex of the projection and from every x-extreme point in the projection of a cell, such that the segments are maximal and contained inside the projection of the cell. We then extend each of these segments into a vertical wall contained inside the original 3D cell. For details and illustrations of vertical decompositions for arrangements of spheres in 3-space, see [4].

Clarkson et al. [4] show that the complexity of the vertical decomposition of an arrangement of spheres is dominated by the complexity of the vertical walls erected from curves in E. They show a slightly super-cubic upper bound on the complexity of the vertical decomposition of an arbitrary collection of n spheres in 3-space. In the next theorem we show another favorable property of the collection of spheres that we study:

Theorem 2.3 *The complexity of the vertical decomposition of the arrangement $\mathcal{A}(M)$ for a collection of spheres M as defined in Theorem 2.1 is $O(n^2)$.*

Proof. We distinguish between two portions of 3-space: inside the union of the balls (i.e., the balls corresponding to the spheres in M), and outside the union of the balls. The complexity of the vertical decomposition of $\mathcal{A}(M)$ inside the union of the balls is evidently $\Theta(n)$, by Theorem 2.1 (i). Hence, from this point on we bound the complexity of the vertical walls that lie completely outside the union of the balls. These walls are erected from curves that lie on the union boundary. Recall that the equators of the spheres are also considered curves in the arrangement $\mathcal{A}(M)$.

Fix an edge γ of $\mathcal{A}(M)$ that lies on the union boundary, and let H_γ be the vertical surface which is the union of vertical lines through points on γ. Let F be the collection of faces of $\mathcal{A}(M)$ that lie on the union boundary, where a face is a maximal portion of a sphere in M that does not meet any other sphere or the equator of that sphere, and is not contained in any other sphere. By Theorem 2.1 (parts (i) and (ii)) F consists of $O(n)$ faces (or surface patches), each bounded by a small number (bounded by a constant) of low-degree algebraic curves.

For every face f in F, let f' be $f \cap H_\gamma$. f' is a collection of a small number of low degree algebraic curves on H_γ. By standard arguments, the complexity of the vertical wall extended from γ and that lies outside the union, is determined by the complexity of the lower envelope (or upper envelope, or both) defined by the collection $F' = \{f'|f \in F\}$ with regard to the curve γ. Since the curves F' are either pairwise disjoint, or a pair of curves in F' meet at an endpoint of at least one of the curves, and they do not intersect otherwise, the complexity of the envelope is $O(n)$. We repeat this argument for every edge γ on the union boundary. There are $O(n)$ such edges, and the upper bound follows. □

The bound is sharp. One can easily construct a set of (even non-intersecting) unit spheres that achieve $\Theta(n^2)$ complexity. It is though possible to construct an $O(n)$ decomposition of the arrangement. Consider a three-dimensional grid whose unit size (along each of the coordinate axes) is $2 \times r_{\max}$. The grid induces a partitioning of 3-space into axis parallel unit cubes (those cubes whose vertices are grid points and that do not contain any grid point in their interior). By applying the same type of arguments as in the proof of Theorem 2.1, one can show that every cube in this partitioning intersects at most some fixed number of spheres. Moreover, each sphere intersects at most 8 cubes. Now take the collection of $O(n)$ non-empty cubes. It is easy to see that the vertical decomposition of this set has linear complexity. For each of the cubes consider the arrangement of spheres that intersect it, restricted to the cube and construct its vertical decomposition (again restricted to the cube). This will give a decomposition of the cube in a bounded number of simple pieces. As there are $O(n)$ cubes to decompose the total complexity will be linear. It is easy to see that the number of adjacencies between pieces is linear as well, which is an important property in certain applications (like motion planning).

Theorem 2.4 *For a collection of spheres M as defined in Theorem 2.1 there exists a decomposition of the arrangement $\mathcal{A}(M)$ into simple pieces of total complexity $O(n)$.*

3 Intersection Queries

Given a molecule it is required that we are able to answer certain queries efficiently. In particular intersection queries play an important role. For example, in computer aided drug design, one wishes to manipulate a molecule in the presence of another molecule to see whether they fit together. During such manipulation the molecules should not intersect. Hence, with every change in the manipulated molecule an intersection query with the other molecule must be performed. Also, it is often required to know whether particular positions in 3-space lie inside or outside the molecule, i.e., whether a point intersects the molecule or not.

In this section we will devise a data structure that can be used to answer intersection queries with either a point or with a ball whose radius is bounded by $r_{\max}$. The query should report all atom balls that contain the point or intersect the query ball. We will use this structure in the next section to compute the molecule boundary. Clearly a point is a degenerated ball. Hence, a data structure that can answer queries for balls can also answer point queries. Below we only describe a structure for query balls. When implementing the structure though a distinction is made because point queries can be answered faster (i.e., with smaller constants in the time bounds).

Our first approach to solving this problem transforms it into an orthogonal range query problem. Let M be the set of n balls (as defined in Theorem 2.1) to store and let C be the set of centers of the balls. We store the set C in a three-dimensional range tree that can answer the following type of query: Given a query axis-parallel box, report the points of C contained in the box. To report all the balls of M intersecting a query ball Q with radius r_Q we construct a ball Q' with radius $r_Q + r_{\max}$ and concentric with Q. Clearly, the center of a ball of M intersecting Q must lie inside Q'. Next, let $\bar{Q}'$ denote the axis-parallel bounding box of Q'. We query the range search structure on C with the box $\bar{Q}'$. Using the same technique as in the proof of Theorem 2.1(i), it can be shown that the number of answers is bounded by a constant. For each of these answers we check whether the corresponding ball actually intersects Q. This takes $O(1)$ time. Using a standard range tree (see, e.g., [18, Section 2.3.4]) we obtain a solution that requires $O(n \log^2 n)$ preprocessing time and space and answers queries in time $O(\log^2 n)$.

However, due to the special properties of the collection of spheres that we study, they admit a simpler and more efficient data structure. Like in the previous section, we subdivide space into cubes of size $2 \times r_{\max}$. For each ball in M we compute the grid cubes that it intersects. Let $\mathcal{C}$ be the set of non-empty grid cubes. The size of $\mathcal{C}$ is bounded by $O(n)$. We arrange the cubes of $\mathcal{C}$ in a balanced binary search tree, ordered by the lexicographic order of the (x, y, z) coordinates of the say, bottom-left-front vertex of the cube. With each non-empty cube we store the list of (at most a constant number of) balls of M that intersect it.

Given a query ball Q, we compute all the (at most 8) grid cubes it intersects, and search for each of these cubes in the binary tree, to see whether it is nonempty. If it exists we check the balls stored for intersection with Q. We might find some balls more than once but duplicates can easily be removed. The total number of balls tested will be $O(1)$. This leads to the following result:

Theorem 3.1 *Given a collection of n balls M as in Theorem 2.1, one can construct a data structure using $O(n)$ space and $O(n \log n)$ preprocessing time, to answer intersection queries for balls whose radii are not greater than $r_{\max}$, in time $O(\log n)$.*

The binary search tree in the above approach is only used to locate the different cubes. Hence, we can easily replace it by a hashing structure. Using the perfect hashing results in [7] this leads to:

Theorem 3.2 *Given a collection M of n balls as defined in Theorem 2.1, one can construct a data structure using $O(n)$ space, to answer intersection queries for balls whose radii are not greater than $r_{\max}$, in $O(1)$ time. The randomized preprocessing time of the structure is $O(n)$.*

Recently, Fjällström and Petersson [8] have carried out a comparative study of the behavior in practice of various three-dimensional range search structures, motivated by simulation of deformation processes. An interesting result of their study is that range trees do not perform well in practice (for the special test data that they experiment with), as compared to other methods that they examined. In particular the best performing method in their study is a grid-like method, similar to the one mentioned above.

We implemented this data structure although we used double hashing rather than perfect hashing. The following table shows, for each of the four test molecules: their size, the building time, the time to answer 1000 point intersection tests and the time to answer 1000 intersection tests for a ball with radius $r_{\max}$. All running times are in seconds, on an Indigo R3000 workstation (25.4 specmarks).

mol.	n	build	point query	ball query
1	42	0.01	0.14	0.20
2	100	0.01	0.13	0.22
3	757	0.08	0.13	0.19
4	971	0.12	0.14	0.22

As can be seen the query time seems independent of the size of the molecule. Also, the query time bounds are very low allowing, e.g., for testing whether a molecule with 100 atoms intersects a molecule with 1000 atoms in 0.02 seconds, i.e., fast enough for interactive use while manipulating the molecule.

4 Computing the Boundary

We now use the data structure of the previous section to efficiently compute the outer cell of the boundary of the union of the spheres. We assume that the union consists of one connected component, which is often the case in our application setting. If the union is composed of several pairwise-disjoint connected components, then additional measures must be taken; see below for more details.

The boundary of the outer cell can be viewed as a formal molecular surface, which is of interest in various applications in molecular biology. Moreover, we will show that the same algorithm, operating on a closely related set of balls, solves another problem in computational biology, sometimes referred to as computing the *approximate solvent accessible surface* (see, e.g., [14, 5, 16]). This involves the interaction of solute and solvent molecules in a solution. In a simplified model, the question may be formulated as follows: Which parts of a molecular surface of a solute molecule are accessible to the solvent molecules, where the latter are modeled by spheres (a single sphere each). We will assume that the solute molecules are modeled by the hard sphere model.

An approach to solving this problem is proposed in [14]: Roll a sphere representing the solvent molecule on a reference surface, to obtain a new surface described by the center of the rolling sphere. The reference surface, in our case, is the boundary of the union of the balls in the hard sphere model. In most cases, the solvent "sphere" is assumed to be fairly small. We denote the solvent molecule sphere by R.

We can rephrase the above problem in terms of *motion planning in robotics*: Let R be a spherical robot moving in 3-space among spherical obstacles (the balls in M). Describe the obstacles in a configuration space where every point represents a possible placement of R by the position of the center of R. In this formulation, the solvent accessible surface becomes the boundary of the outer cell of the free portion of the configuration space. Unlike prevailing formulations of motion-planning problems, the "obstacles" in our case may intersect. However, as expressed in Theorem 2.1, these obstacles have other, favorable, properties.

To compute the boundary of the outer cell of the free configuration space, we follow a common practice in motion planning, and compute the Minkowski (vector) difference[1] of each obstacle and the robot, to obtain the configuration obstacles. Now, the Minkowski difference of two spheres is again a (larger) sphere. Hence, our goal is actually to compute the outer component of the union boundary for a collection of balls in 3-space.

The solute molecule is modeled by balls as in Theorem 2.1. The radius of each ball in M is increased by r'—the radius of the solvent sphere. We assume r' to be of the same order of magnitude as $r_{\max}$. Note that, the expanded balls obey the conditions of Theorem 2.1 for different constants $r'_{\min}$, $r'_{\max}$, k' and ρ'. Therefore, an algorithm for computing the union boundary for a collection M of balls as in Theorem 2.1, is also applicable to computing solvent accessible surfaces. Thus, we present an algorithm for computing the union boundary of a collection of balls M as defined in Theorem 2.1.

The scheme of the algorithm is to compute, for each $B_i \in M$, the contribution of its boundary to the union boundary and then to combine all this information to give the final output of the algorithm. The algorithm consists of four steps:

1. For each ball identify the other balls intersecting it.

2. For each ball compute its (potentially null) contribution to the union boundary.

3. Transform the local information into global structures describing the connected components of the union boundary.

4. Filter out portions of the union boundary that do not belong to the desired connected component of the "outer cell".

[1] The Minkowski difference of two spatial sets A and B, $A - B$, is the set $\{p - q | p \in A, q \in B\}$.

If all we need is a list of the faces of the union boundary (as is the case in the preparatory stage of the hidden surface removal algorithm of Section 5) then we only need Steps 1 and 2.

For Step 1, we use the data structure described in Theorem 3.2. The cost of the query for each ball is $O(1)$, hence the total cost of this step is $O(n)$ randomized time (or $O(n \log n)$ deterministic).

As for Step 2, consider a ball B_i and the family of (a constant number of) balls intersecting it. Let B_j be a ball intersecting B_i. If B_j fully contains B_i, then we stop the process for B_i, as it cannot contribute in any way to the union boundary. If B_j is fully contained in B_i, then we ignore B_j, for obvious reasons. Otherwise, we compute the intersection between the spheres S_i and S_j, which is a circle C_{ij} on S_i. The circle C_{ij} partitions S_i into two parts: One part may appear on the union boundary, and we will refer to it as the *free* part of S_i with respect to S_j, and the other is completely contained in S_j and therefore cannot appear on the union boundary. We repeat the process for each ball intersecting B_i, to get a collection of circles on S_i. These circles form a 2D arrangement $\mathcal{A}_i$ on S_i. A face of $\mathcal{A}_i$ belongs to the union boundary if and only if it belongs to the free portion defined by each circle C_{ij}. Since the number of circles on S_i defined in that way is constant, the arrangement can be computed by a brute force method in constant time, together with the attribute whether a face is free or not (a free face is guaranteed to appear on the union boundary). The complexity of the arrangement $\mathcal{A}_i$ is evidently constant. For each ball in M, the above procedure will take constant time.

In Step 3 we will represent each connected component of the union boundary by a *quad-edge* structure [10]. To this end we have to augment the arrangements $\mathcal{A}_i$ slightly. If $\mathcal{A}_i$ is the whole sphere S_i we split it into two parts with some circle. Next, if a boundary component of $\mathcal{A}_i$ is a simple circle C we split C into two arcs adding two vertices. (If C lies in both $\mathcal{A}_i$ and $\mathcal{A}_j$ the same vertices should be added in both arrangements.) Finally, if a free face of $\mathcal{A}_i$ contains holes we split it by adding extra arcs. (To make these additions canonical, we can fix a direction d, and add all the extra arcs along great circles that are intersection of the sphere with planes parallel to the direction d.) After this step, which can easily be performed in time $O(n)$, the union boundary will consist of simple faces where each face is bounded by at least two edges and each edge bounds exactly two faces (assuming general position).

Now we proceed as follows: We take some free face f of some $\mathcal{A}_i$, that was not treated before, and determine its boundary. In this way we obtain for each edge e bounding f pointers to the previous and next edge along f, as required for the quad-edge structure. Also for each edge e bounding f we determine the arrangement $\mathcal{A}_j$ containing the face f' on the opposite side of f (j can be i because of the extra arcs we added). f' can be found in time $O(1)$ because only a constant number of spheres intersect S_i. We locate e in $\mathcal{A}_j$, add pointers between the copies of e and recursively treat f' in $\mathcal{A}_j$ if it was not visited before. In this way we continue until we have located the whole connected component of the boundary containing f. If not all free faces have been visited we take one of the non-visited faces and continue from there. It is easy to see that this requires time $O(n)$ in total.

As a result of Steps 2 and 3 we get a quad-edge representation of each connected component of the union boundary of the balls in M. Recall that we are only interested in the boundary of the outer cell, which we assumed to be connected. At this stage we need to identify the outer cell (Step 4). We search for the ball $B_l \in M$ having the point with largest z-coordinate on its boundary. We then determine an edge on the boundary of the face f_l of B_l containing the highest point. This face clearly belongs to the outer cell. From this face we scan the entire connected component containing f_l, using the quad-edge structure. We mark every edge found throughout the scan, as belonging to the outer cell. Finally we delete any edge that has not been marked in the previous step. The remaining structure describes the outer cell. This takes time proportional to the size of the union boundary which is $O(n)$.

As for space requirements, the data structure of Theorem 3.2 requires $O(n)$ space. The additional structures described above are easily verified to require linear space.

Theorem 4.1 *The outer portion of the boundary of the union of a connected collection of balls as defined in Theorem 2.1 can be computed in $O(n)$ randomized time (or $O(n \log n)$ deterministic), using $O(n)$ space.*

If the union of the balls is not connected, additional machinery is required to identify and discard components lying in internal cells. This can be done in time $O(n \log n)$ by computing the decomposition of $\mathcal{A}(M)$ as described in Theorem 2.4 and traversing the outer cell of this arrangement. Details will be given in the full paper.

We only implemented Steps 1 and 2 of the above algorithm because that is all that is required for hidden surface removal. The following table shows the running time for our four molecules.

mol.	Step 1	Step 2
1	0.02	0.2
2	0.05	0.7
3	0.42	6.5
4	0.54	7.5

As can be seen Step 2 dominates the amount of time required, while theoretically both steps take time $O(n)$. It is though easy to verify that the operations involved in Step 2 are much more complicated and, hence, the constants in the bound are much higher.

5 Hidden Surface Removal

One of the tasks of molecular modelling packages is to display molecules. This should preferably be done so fast that the user can interact with the model, e.g., by turning it around to look at it from different directions or by moving different molecules with respect to each other, for example to see whether they fit together in some nice way. Hence, one needs fast algorithms for hidden surface removal among sets of intersecting spheres.

In practice one normally uses the *Z-buffer algorithm* for this. Almost all 3-D graphics workstation available nowadays have such an algorithm implemented in their graphics hardware. Unfortunately, such hardware can only handle polyhedral objects. Hence, as a first step, one has to approximate the spheres by triangular meshes. Different methods exist but all lead to a large number of faces. To get a reasonable sphere one needs at least 100 triangles. So a molecule of 1000 atoms requires the drawing of 100,000 triangles in 3-space. Even special graphics hardware (except maybe the most expensive types) cannot display such molecules fast. It would typically take between one and a few seconds.

A second approach used in computer graphics is the *painter algorithm*. Here one tries to define a depth order on the objects, sorting them from back to front. Next one draws the objects in this order on top of each other (like a painter) where each new object hides the parts of other objects that lie below it. Such an approach does not require special graphics hardware and, hence, can also be used on, e.g., laser printers. The problem though is that one needs a valid depth order on the objects. Such an order does not always exist (there can be cyclic overlap among objects). Also, for intersecting object such an order obviously does not exist.

For sets of non-intersecting spheres an easy depth order exists: simply sort the spheres by z-coordinate of their center. But for intersecting spheres this does not apply. There is though no need to draw the entire spheres. It is good enough to draw the pieces of the spheres that constitute the boundary of the union, as computed in the previous section. Clearly, pieces of the spheres that do not contribute to the boundary cannot be visible. Also the parts of the sphere that point away from the viewing direction cannot be seen. We will show that for the remaining part of the boundary a depth order does exist.

Theorem 5.1 *Let $S = S_1, ..., S_n$ be a collection of spheres sorted by decreasing z-coordinate of their center (i.e., from back to front). For each S_i let H_i be the hemi-sphere facing the viewing direction. Let $H_i^1, ..., H_i^{j_i}$ be the collection of maximal pieces of H_i that are part of the boundary of the union of the spheres. Then*

$$H_1^1, ..., H_1^{j_1}, H_2^1, ..., H_2^{j_2}, ..., H_n^1, ..., H_n^{j_n}$$

is a valid depth order for the pieces on the boundary.

Proof. We will prove this by contradiction. So assume there are two pieces H_i^k and H_j^l that are in the wrong order. Clearly $i \neq j$ so let us assume $i < j$. H_i^k and H_j^l are in the wrong order so there must be a ray in the viewing direction that first hits H_i^k and than H_j^l (so H_i^k partially hides H_j^l). Because H_i^k and H_j^l belong to the union boundary, the ray must first hit H_i^k, next the back of H_i, next H_j^l, and finally the back of H_j. But this implies that the back of H_j lies behind the back of H_i and, hence, $j < i$, which contradicts our assumption. □

This leads to the following algorithm. We first compute the boundary of the union. In this way, for each sphere S_i we collect a (constant) number of pieces that it contributes to the boundary. For each of these pieces we cut off the part that does not lie in H_i. Next we sort the spheres by depth and draw the pieces in this order. As shown in the previous section computing the boundary takes deterministic time $O(n \log n)$ and results in $O(n)$ pieces. The sorting of the spheres by depth takes time $O(n \log n)$. Hence, we obtain the following result:

Theorem 5.2 *Given a set of n spheres as defined in Theorem 2.1, a valid depth order consisting of $O(n)$ pieces can be computed in time $O(n \log n)$.*

We implemented this algorithm. Actually, we split it into two phases: In the preprocessing phase we compute the boundary of the union and for each sphere we collect the pieces that are part of the boundary. (See the previous section for time bounds.) In the query phase we are given a particular viewing direction. Now we compute the hemi-spheres (which are dependent on the viewing direction) and cut off the parts that lie outside it. Next we compute the depth order for the spheres and draw the pieces. All pictures in this paper were generated using this approach. The implementation (even though not yet fully optimized) is rather fast. We did run it on two different machines: An Indigo R3000 machine with entry level graphics and an Indigo R3000 XS24 with hardware Z-buffer. The first machine performs the Z-buffer algorithm in software while the second one has special purpose hardware for this. The following table shows the amount of time (in seconds) required per frame for both the Z-buffer algorithm and our method.

	entry graphics		XS-24	
mol.	Z-buf	new	Z-buf	new
1	0.5	0.2	0.1	0.1
2	1.1	0.5	0.25	0.3
3	5.5	3.6	1.85	2.0
4	7.4	4.6	2.35	2.9

From these figures one might conclude that the new method runs about twice as fast on the entry level system while it is slightly slower on the XS-24. This should though be treated with care. First of all, the pictures produced by the two methods are quite different. The way we implemented it, the Z-buffer algorithm produces a Gouraud shaded image while our new method produces a flat shaded image (because the method draws in 2-d one cannot produce anything else). Our method though draws boundary lines around the spheres and along the intersections (like in Figures 1 till 4) while the Z-buffer method does not (it cannot because it does not know where the intersections are). Also the picture quality differs largely. On 8-bit systems the Gouraud shaded image looks rather displeasing while the images produced by the new method look much nicer. Also there is the issue of precision. For the Z-buffer we used an approximation of the spheres with about 100 triangles. In the new method we used an approximation of the circles and ellipses (projected sphere boundaries and intersections) with 24 edges, which is better. Finally, the difference in time for our new method on the two different platforms is caused because, on the entry level systems, more than half of the time is spent in the actual drawing, not in the computation of the depth order. On the XS-24 this is much less. For more details on the implementation see [9].

So we can conclude that our new approach produces nice images and can be used on simple graphics workstations without 3-D capabilities. Even on machines with 3-D graphics hardware the method behaves comparably. Finally, and probably most important, the new method can be used to produce high quality Postscript pictures which is impossible with the Z-buffer algorithm.

In some applications a depth order is not enough but one wants a combinatorial representation of the visible pieces, the *visibility map.* A lot of work has been done in computational geometry in producing visibility maps. See, e.g., the book of de Berg [6]. For collections of arbitrary intersecting spheres the best known hidden surface removal algorithm runs in randomized expected time $O(n^{2+\epsilon})$ for any $\epsilon > 0$ (using the general algorithm for computing the lower envelope of low-degree algebraic surfaces in 3-space by Sharir[19]). No output-sensitive method, where the time bound depends on the complexity of the resulting visibility map, is known. In the case of a set of non-intersecting spheres the best known method is presented in [12]. It runs in time $O((n+k)\log^2 n)$ where k is the complexity of the visibility map (which can be $\Theta(n^2)$, even for non-intersecting unit spheres). We will now briefly indicate how the method of [12] can be adapted to work for intersecting spheres as well. For more details see the full paper. We assume some familiarity with the method of [12].

For the method of [12] to work one needs to subdivide the objects in a (small) collection of pieces that first of all has a depth order and, secondly, has the property that the union of the projection on the viewing plane of the pieces lying in any depth range has small complexity. One could use the collection of pieces as defined in Theorem 5.1. It is though unclear whether this collection satisfies the second condition. Hence, we define a different set of pieces.

For each sphere S_i we again consider the hemisphere H_i. We take the pieces of H_i that are contained in the boundary of the union of $\{H_1, ..., H_i\}$, i.e., we ignore the spheres that lie nearer. In a way similar to the proof of Theorem 5.1 one can prove that these pieces constitute a valid depth order. In the full paper we will also prove that for any z-range the union of the projection of the pieces whose sphere center lies in the range, is linear. Applying the results in [12] we obtain:

Theorem 5.3 *Given a set of* n *spheres as defined in Theorem 2.1, the visibility map can be computed in time* $O((n+k)\log^2 n)$*, where* k *is the complexity of the resulting map.*

6 Conclusion

We have considered the hard sphere model of a molecule in a fixed nuclear configuration from a computational geometry point of view. We have shown that the collection of spheres representing the atoms of a molecule in that model behaves favorably in comparison with an arbitrary collection of spheres in 3-space. Using this observation we were able to devise a data structure that efficiently detects which atom spheres of a molecule are intersected by a query sphere (of bounded radius). Then, we presented an efficient algorithm for computing the boundary (or envelope) of the union of atom spheres. Furthermore, we presented efficient algorithms for hidden surface removal of a molecule. Our results have been supported by experiments carried out on actual protein data, displaying the efficiency and practicality of our approach.

We believe that other (algorithmic) problems in molecular modelling can be solved efficiently using the data structures and properties presented in this paper. The area also provides many other challenging problems, like: Is it possible to preprocess two molecules such that one can efficiently determine whether, in a particular pose, the molecules intersect or not? Also other models of molecules give rise to interesting question. For example, there is the problem of hidden surface removal for the ball-and-stick model where atoms are represented by small balls and the links by thin sticks. We plan to work further on these problems.

Acknowledgements

The authors wish to thank Ed Moret, Raquel Norel and Haim Wolfson for helpful discussions and for supplying various data and references on molecular structures. We would like to thank Geert-Jan Giezeman who did most of the implementation work.

References

[1] E.E. Abola, F.C. Bernstein, S.H. Bryant, T.F. Koetzle and J. Weng, Protein data bank, in *Crystallographic Databases: Information Content, Software Systems, Scientific Applications*, F.H. Allen, G. Bergerhoff and R. Seivers, Eds., Data Comission of the International Union of Crystallography, Bonn/Cambridge/Chester, 1987, pp. 107–132.

[2] F.C. Bernstein, T.F. Koetzle, G.J.B. Williams, E.F. Meyer Jr., M.D. Brice, J.R. Rodgers, O. Kennard, T. Shimanouchi and M. Tasumi, The protein data bank: A computer-based archival file for macromolecular structure, *Journal of Molecular Biology* **112** (1977), pp. 535–542.

[3] B. Chazelle, H. Edelsbrunner, L. Guibas and M. Sharir, A singly-exponential stratification scheme for real semi-algebraic varieties and its applications, *Theoretical Computer Science* **84** (1991), pp. 77–105.

[4] K.L. Clarkson, H. Edelsbrunner, L.J. Guibas, M. Sharir and E. Welzl, Combinatorial complexity bounds for arrangements of curves and spheres, *Discrete and Computational Geometry* **5** (1990), pp. 99–160.

[5] M.L. Connolly, Solvent-accessible surfaces of proteins and nucleic acids, *Science* **221** (1983), pp. 709–713.

[6] M. de Berg, *Ray shooting, depth orders and hidden surface removal*, Springer LNCS 703, Springer-Verlag, 1993.

[7] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert and R.E. Tarjan, Dynamic perfect hashing: upper and lower bounds, *Proc. 29th IEEE Symposium on Foundations of Computer Science*, 1988, pp. 524–531.

[8] P.-O. Fjällström and J. Petersson, Evaluation of algorithms for geometrical contact-searching problems, *manuscript*, 1993.

[9] G-J. Giezeman and M.H. Overmars, Fast display of molecular models, in preparation.

[10] L.J. Guibas and J. Stolfi, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *ACM Transactions on Graphics*, **4** (1985), pp. 74–123.

[11] *Handbook of Biochemistry*, H.A. Sober, Editor, The Chemical Rubber Co., 2nd Edition, 1970.

[12] M.J. Katz, M H. Overmars and M. Sharir, Efficient hidden surface removal for objects with small union size, *Comp. Geom.: Theory and Appl.*, **2** (1992), pp. 223–234.

[13] K. Kedem, R. Livne, J. Pach and M. Sharir, On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles, *Discrete and Computational Geometry* **1** (1986), pp. 59–71.

[14] B. Lee and F.M. Richards, The interpretation of protein structure: Estimation of static accessibility, *J. of Molecular Biology* **55** (1971) pp. 379–400.

[15] J. Matoušek, N. Miller, J. Pach, M. Sharir, S. Sifrony and E. Welzl, Fat triangles determine linearly many holes, *Proc. 32nd IEEE Symposium on Foundations of Computer Science*, 1991, pp. 49–58.

[16] P.G. Mezey, Molecular surfaces, in *Reviews in Computational Chemistry*, Vol. I, K.B. Lipkowitz and D.B. Boyd, Eds., VCH Publishers, 1990.

[17] M. H. Overmars, Point location in fat subdivisions, *Inform. Proc. Lett.*, **44** (1992), pp. 261 – 265.

[18] F.P. Preparata and M.I. Shamos, *Computational Geometry—An Introduction*, Springer Verlag, New York, 1985.

[19] M. Sharir, Almost tight upper bounds for lower envelopes in higher dimensions, *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci.*, 1993, pp. 498–507.

[20] A.F. van der Stappen, D. Halperin and M. H. Overmars, The complexity of the free space for a robot moving amidst fat obstacles, *Comput. Geom.: Theory and Appl.*, **3** (1993), pp. 353–373.

[21] A. Varshney and F.P. Brooks Jr., Fast analytical computation of Richards's smooth molecular surface, *Proc. Visualization '93,* 1993, pp. 300–307.

Determining the Castability of Simple Polyhedra*

Prosenjit Bose
McGill University

David Bremner
McGill University

Marc van Kreveld
Utrecht University

Abstract

A polyhedron P is castable if its boundary can be partitioned by a plane into two polyhedral terrains. Such polyhedra can be manufactured easily using two cast parts. Assuming that the cast parts are removed by a single translation each, it is shown that for a simple polyhedron with n vertices, castability can be decided in $O(n^2 \log n)$ time and linear space using a simple algorithm. Furthermore, a more complicated algorithm solves the problem in $O(n^{3/2+\epsilon})$ time and space, for any fixed $\epsilon > 0$. In the case where the cast parts are to be removed in opposite directions, a simple $O(n^2)$ time algorithm is presented. Finally, if the object is a convex polyhedron and the cast parts are to be removed in opposite directions, a simple $O(n \log^2 n)$ algorithm is presented.

1 Introduction

A new application area of computational geometry is in the area of automated manufacturing, where an engineer can design an object with the aid of a computer, and determine by which manufacturing process the object can be constructed. There are several types of manufacturing processes studied in computational geometry, such as injection molding [5, 6, 13], NC machining [14], automated welding [17] and layer deposition methods (such as stereolithography) [3].

In this paper, we study the geometric and computational aspects of casting. Casting consists of filling the open region bounded by two or more cast parts with a material such as a liquid metal, after which the cast parts are removed. The removal of the cast parts without breaking them imposes certain restrictions on the shape of the object to be constructed. For *sand casting* (see e.g. [12, 23]), only two cast parts are used. To construct the cast parts, a prototype of the object is first obtained (see Figure 1). The prototype is then divided into two parts along a plane. The face along which the part is cut is referred to as the base. The first cast part is made by placing the base of the part on a flat surface, and then adding sand around it. The part is then rotated such that the base is facing up. The other part is placed such that the bases coincide and the second cast part is built by adding sand around this part. An opening into the cavity is maintained during the construction of the second cast part. This completes the construction of the cast of the prototype object. To built a metal rendition of the prototype object with this cast, liquid metal is poured into the opening until it fills the cavity. After the metal solidifies, the cast parts are removed from the object. To be able to construct the cast with this process, it is necessary that the prototype can be removed without breaking the cast. Also, for manufacturing methods related to sand casting such as other metal casting methods [12, 23] and injection molding and blow molding methods for plastics [20, 24], it must be possible to remove the constructed object from the cast without breaking it in order to re-use the cast. Thus for several different manufacturing methods involving casting, the geometry of the object determines its feasibility of construction.

We note that more complicated objects can be made by using cores and inserts [12, 20, 23, 24]. However, their use slows down the manufacturing process and makes it more costly. Thus to be cost efficient, cores and inserts should be avoided. We do not study the extra possibilities of cores and inserts in this paper.

An object is *castable* if it can be manufactured by casting. Geometric and algorithmic issues of casting of planar objects has been studied by Rosenbloom and Rappaport [21]. This paper addresses casting of polyhedral objects. In geometric terms, castability can be defined as follows (for a polyhedron P, ∂P denotes the boundary of P, and for a plane h, h^+ and h^- refer to the open half-spaces above and below h):

Definition 1 *A simple polyhedron P is* castable *if there exists a plane h such that $h^+ \cap \partial P$ is a weak terrain in some orientation, and $h^- \cap \partial P$ is a weak terrain in some*

[*] This research is supported by the NSERC and the ESPRIT Basic Research Action 7141 (project ALCOM II: *Algorithms and Complexity*).

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

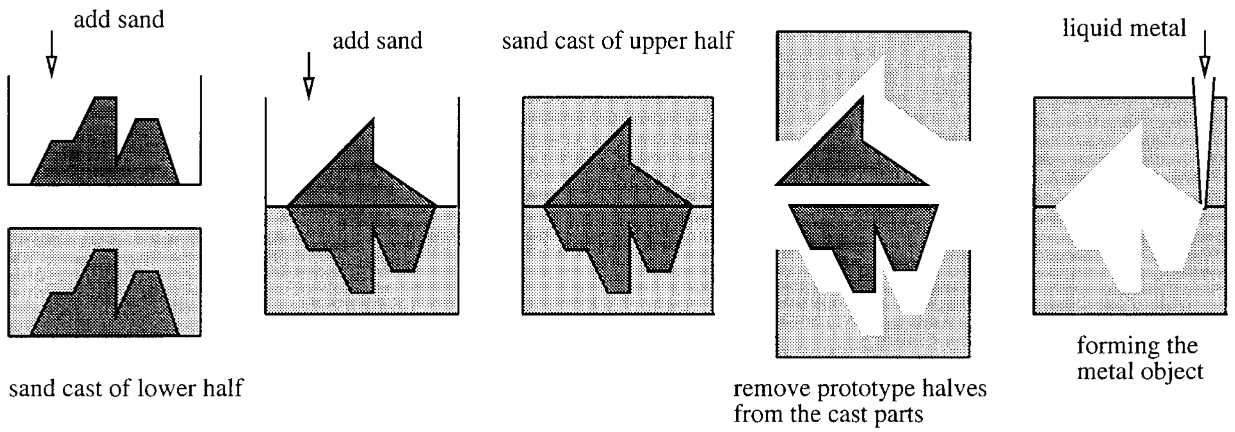


Figure 1: Construction of an object by sand casting, using two halves of the object as prototypes.

orientation. The plane h is called the casting plane. *(A weak terrain may contain edges and facets parallel to the orientation in which it is a terrain.)*

To manufacture a polyhedron P that is castable, first determine a casting plane h for P. Then the cast parts C_1 and C_2 are made from the prototype halves $h^+ \cap \partial P$ and $h^- \cap \partial P$. Since P is castable, the prototype halves can be removed from the cast parts, and later the manufactured object can be removed from the cast parts. Although our algorithms deal with the sand casting process, they can be applied in related processes as well. We consider three versions of the castability problem. They differ in the way the cast may be removed from the polyhedron P. Figure 2 shows the three versions for planar polygons.

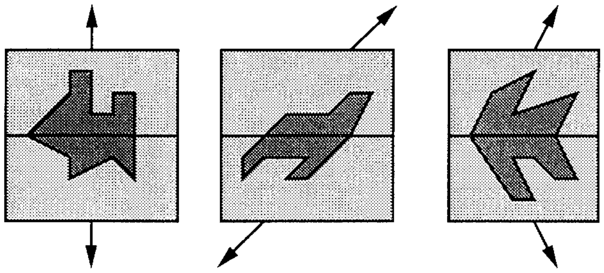


Figure 2: Three versions of the castability problem.

1. The two cast parts must be removed from P by one translation each, in opposite directions, and normal to the casting plane (orthogonal cast removal).
2. The two cast parts must be removed from P by one translation each, and in opposite directions (opposite cast removal).
3. The two cast parts must be removed from P by one translation each, in arbitrary directions (arbitrary cast removal).

In manufacturing, developing machines that perform orthogonal and opposite cast removal is much simpler than machines that perform arbitrary cast removal. In fact, opposite cast removal seems to be the most popular technique used [8, 20]. Furthermore, if orthogonal or opposite cast removal is possible, it can be determined more efficiently. We summarize the complexity of the different algorithms we developed for the casting problem. In the top half of the table, the time bounds of simple, linear space algorithms are shown. The bottom half of the table shows improvements made (in theory) by using $O(n^{3/2+\epsilon})$ storage (for any positive constant ϵ).

lin. space	orthogonal	opposite	arbitrary
convex	$O(n \log^2 n)$	$O(n \log^2 n)$	$O(n^2 \log n)$
simple	$O(n^2)$	$O(n^2)$	$O(n^2 \log n)$

best res.	orthogonal	opposite	arbitrary
convex			$O(n^{3/2+\epsilon})$
simple	$O(n^{3/2+\epsilon})$	$O(n^{3/2+\epsilon})$	$O(n^{3/2+\epsilon})$

Our algorithms depend on geometric properties of casting, discussed in Section 2. The maximum number of distinct cast planes is analysed in Section 3, and algorithms are given in Sections 4 and 5. For detailed proofs and descriptions of the algorithms, we refer the reader to the technical report [4].

2 Preliminaries

For a polyhedron P, denote by V, E and F the set of vertices, edges and facets. Edges that bound two parallel facets are not allowed; they can be removed without changing the shape of the polyhedron. Edges and facets are open, the closure of an edge e or facet f is denoted by $cl(e)$ or $cl(f)$. Although technically the object to be constructed is the interior of P, and the boundary of P is part of the cast, with a slight abuse of notation, we nevertheless state that P is castable or not castable.

For a non-vertical plane h, we denote by h^+ and h^- the open half-spaces above and below h. If h is vertical but does not contain a line parallel to the y-axis, then h^+ and h^- denote the open half-spaces bounded by h that contain the points $(0, \infty, 0)$ and $(0, -\infty, 0)$, respectively. If h is vertical and contains a line parallel to the y-axis then h^+ and h^- denote the open half-spaces bounded by h that contain the points $(\infty, 0, 0)$ and $(-\infty, 0, 0)$, respectively.

Let P be a polyhedron and let h be a plane that intersects P. If h is a casting plane and the upper cast part can be removed in direction $\vec{u}$, then the outward normal of any facet f of P that lies in h^+ makes an angle at most $\pi/2$ with $\vec{u}$. This also holds when f lies partially in h^+. Also, if all facets of P that lie completely or partially in h^+ make an angle at most $\pi/2$ with $\vec{u}$, then $h^+ \cap \partial P$ is a weak terrain. Therefore, castability with respect to a plane h is only determined by the facets of P that intersect h^+ and the ones that intersect h^-. If h is a casting plane for P, then h can be perturbed if this does not involve new facets intersecting h. In case of orthogonal cast removal, the only perturbation allowed is translation.

Observation 1 *For castability with orthogonal cast removal, we may assume that the casting plane contains at least one vertex of P. For opposite and arbitrary cast removal, we may assume that the casting plane contains at least three vertices of P.*

2.1 Relation to linear programming

Let P be a polyhedron and let h be a plane. The plane h partitions the set V of vertices of P into three subsets V_h, V_h^+ and V_h^- of vertices in, above and below h, respectively. Similarly, h partitions the set E of edges of P in four subsets $E_h^{\subset}$, $E_h^{\times}$, E_h^+ and E_h^- of edges contained in h, intersecting h, above h and below h, respectively. The set F of facets is partitioned in the same way. For any facet $f \in F$, denote by $\Psi(f)$ the closed half-space supporting f but not containing the interior of P. Denote by $\Psi_0(f)$ the same half-space, but translated such that the bounding plane contains the origin. Let

$$\xi^+(h) = \bigcap_{f \in F_h^+ \cup F_h^\times} \Psi_0(f) \quad \text{and} \quad \xi^-(h) = \bigcap_{f \in F_h^- \cup F_h^\times} \Psi_0(f).$$

The intersection of a set of half-spaces is *non-trivial* if it contains more than a single point. Denote by $\mathit{refl}(b)$ the reflection of an object b through the origin. We make the following observations.

Lemma 1 *h is a casting plane for polyhedron P for arbitrary cast removal if and only if $\xi^+(h)$ and $\xi^-(h)$ are both non-trivial.*

Lemma 2 *h is a casting plane for polyhedron P for opposite cast removal if and only if $\xi^+(h) \cap \mathit{refl}(\xi^-(h))$ is non-trivial.*

Lemma 3 *Let h be a plane and let ℓ be a line perpendicular to h and through the origin. h is a casting plane for polyhedron P for orthogonal cast removal if and only if $\ell \cap \xi^+(h) \cap \mathit{refl}(\xi^-(h))$ is non-trivial.*

With the above lemmas, we can test for a given plane h efficiently whether it is a casting plane for P. Since the casting problem for a plane h and a polyhedron P can be transformed in linear time to a linear programming problem in 3 dimensions, the test requires only linear time [18]. In fact, since all planes bounding the half-spaces contain the origin, the linear programming problem is basically 2-dimensional.

Lemma 4 *Given a polyhedron P and a plane h, one can test in linear time whether h is a casting plane for P in any of the three versions for removing the cast.*

2.2 Geometric properties of casting

In this section, we uncover some important geometric properties of casting planes that will allow us to bound the number of distinct casting planes and hence develop efficient algorithms.

Lemma 5 *If a simple polyhedron P is castable in non-opposite directions with casting plane h, then h contains an edge of P.*

Lemma 6 *If a simple polyhedron P is castable with casting plane h and in non-opposite directions, then h contains an edge of the convex hull of P.*

Notice that the above two lemmas imply that if a polyhedron is castable, but not with opposite cast removal, then the casting plane contains both an edge of P and an edge of the convex hull of P (this might be the same edge). This will aid considerably to determine castability with arbitrary cast removal.

The following lemma forms the crucial link between simple polyhedra and convex polyhedra in terms of castability with removal directions restricted to orthogonal and opposite.

Lemma 7 *If a simple polyhedron P is castable, then the convex hull of P is also castable using the same casting plane and cast removal directions.*

Lemma 8 *If the casting plane h properly intersects a facet f of convex polyheron P, and also two vertices u, v in the closure of f, then for opposite cast removal, vertices u, v must be antipodal in $cl(f)$.*

Corollary 1 *Let h be a casting plane for a convex polyhedron P which intersects a facet f properly, and assume opposite cast removal. If h intersects a vertex v and properly intersects an edge e in the closure of f, then v is antipodal to both endpoints of e. If h properly intersects two edges in the closure of f, then they are parallel.*

Lemma 8 and Corollary 1 are the two geometric facts that allow us to solve opposite cast removal more efficiently than arbitrary cast removal.

In [4], we also show that for a convex polyhedron P, the linear programming problems defined by P and a candidate casting plane h need not consider all facets of F, but only those intersecting h and those adjacent to h, leading to the algorithm described in Section 4.

3 The number of distinct casting planes

Given a polyhedron P, two planes h_1 and h_2 are *(combinatorially) distinct* if the partitioning of the facets into F^+, F^-, $F^\subset$ and $F^\times$ they define is different. By Observation 1, a trivial upper bound on the number of distinct casting planes for a polyhedron with n vertices is $O(n^3)$.

This section gives a linear upper bound on the maximum number of distinct casting planes for convex polyhedra in case of orthogonal and opposite cast removal as well as a quadratic upper bound for arbitrary cast removal.

3.1 Orthogonal and opposite cast removal

We restrict our attention to opposite cast removal, since orthogonal cast removal is (close to) a special case of opposite cast removal, and there are only minor differences between the two cases. For an explicit rendition of the differences, we refer the reader to [4]. For opposite cast removal, we have the following properties of intersections of a casting plane h and a polyhedron P: (i) The facets of $F^\times$ that intersect h properly have their outward normals such that when translated to the origin, they span a plane or part of it (since $\bigcap\{\Psi_0(f) \mid f \in F^\times\}$ contains a line through o). (ii) All edges that intersect h properly are parallel (otherwise the incident facets span more than a plane). (iii) Any two vertices in the closure of a facet f and in h are antipodal in $cl(f)$. Any vertex and edge in the closure of f and intersecting h are antipodal in $cl(f)$ (see Lemma 8).

Lemma 9 *Given a convex polyhedron P, the number of distinct casting planes that intersect some edge of P properly is at most linear in the number of vertices of P, assuming opposite cast removal.*

Let h be a candidate casting plane of P, and let $Q = h \cap P$. If Q contains three consecutive vertices u, v, w that are also vertices of P, then each of u and w is either an endpoint of an edge incident to v, or a vertex antipodal to v on the closure of a facet f incident to v. We say that the plane through u, v, w is *generated by* v. It follows that the set of candidate casting planes generated by v has size $\binom{d+a}{2}$, where d is the degree of v and a is the number of vertices antipodal to v in the closures of the facets incident to v. Every casting plane h that does not intersect any edge properly contains three vertices that are consecutive in $h \cap P$, and therefore, every such casting plane is generated by some vertex of P. It is straightforward to prove that every convex polyhedron has a vertex of constant degree which participates in a constant number of antipodal pairs (on the incident facets). Using this fact inductively, we obtain:

Theorem 1 *Given a convex polyhedron P with n vertices, the maximum number of distinct casting planes for P is $O(n)$, assuming opposite removal of the cast parts.*

There is another interesting combinatorial bound on the complexity of the intersection of all distinct casting planes with a convex polyhedron. Two distinct casting planes h_1 and h_2 that properly intersect the same edge are *similar*, because they define the same cast removal directions, and they intersect the same closure of edges and facets. In other words, if h_1 and h_2 each properly intersect edges that are parallel, there cannot be two vertices u, v strictly to the one side of h_1 and strictly to different sides of h_2. We use the term *weakly equivalent* for two such planes. Two planes are *weakly distinct* if they are not weakly equivalent. There are $O(n)$ weakly distinct casting planes for any convex polyhedron P with n vertices. We analyse the combinatorial complexity of $h \cap P$, summed over all weakly distinct casting planes h. This quantity is well-defined for opposite cast removal, since two weakly equivalent casting planes have an equal-size intersection with P (although they may intersect different facets, edges and vertices). We prove a bound of $O(n \log n)$ on the summed complexity. Note that when the sum is over all distinct casting planes (not *weakly* distinct), the summed complexity can be $\Theta(n^2)$ if P has a set of $\Omega(n)$ parallel edges. The bound makes use of a hierarchical decomposition of P that closely resembles the hierarchy of Dobkin and Kirkpatrick [10]. It is the basis of the $O(n \log^2 n)$ time algorithm for casting of convex polyhedra with opposite cast removal.

Lemma 10 *Given a convex polyhedron P with n vertices, there exists a subset V' of the vertices V of size $\Omega(n)$, such that each $v \in V'$ has degree ≤ 8 and is antipodal to ≤ 12 vertices in facets incident to v.*

The following hierarchical decomposition of P generates a set of planes that contains all the candidate casting planes that do not intersect an edge properly:

1. Mark all edges of P as "original". Compute all antipodal pairs of all facets of P and mark them as "original". Set $i = 1$.
2. Select a subset V_i of V as in Lemma 10. For every vertex $v \in V_i$, generate all planes through u, v, w for which $\overline{uv}$ and $\overline{vw}$ are original edges, or (u, v) and (v, w) are original antipodal pairs, or one is an original edge and the other an original antipodal pair. For every vertex $v \in V_i$, the number of generated planes is at most $\binom{12+8}{2} = 190$, thus $O(n)$ for the whole subset.
3. Recompute the convex hull of the vertices of P minus the vertices of V_i.
4. Recompute the antipodal pairs of the facets. (Note: Edges or antipodal pairs of $CH(P \setminus V_i)$ are not original unless they were deemed original in step 1.)
5. Repeat at step 2 with $i = i + 1$ unless P has no vertices left.

As in [10], the number of subsets V_i is $O(\log n)$. It follows—as we show in [4]—that every $\Omega(\log n)$ consecutive vertices of P contained in a casting plane h contains a vertex that generates h. This leads to:

Theorem 2 *Given a convex polyhedron P with n vertices, the maximum total complexity of $h \cap P$, summed over all weakly distinct casting planes h for P, is $O(n \log n)$, assuming opposite cast removal.*

Corollary 2 *Given a convex polyhedron P with n vertices, the maximum number of planes that intersect the interior of P but do not intersect any facets of P is $O(n)$, and the maximum number of edges of P contained in these planes, summed over all planes, is $O(n \log n)$.*

3.2 Arbitrary cast removal

We have shown that the number of casting planes that also allow opposite cast removal is linear. For the other casting planes, we know from Lemma 5 that they contain an edge of P. Since we may also assume that they contain a third vertex, we conclude:

Theorem 3 *Given a convex polyhedron P with n vertices, the maximum number of distinct casting planes for P is $O(n^2)$, assuming arbitrary removal of the cast parts.*

4 Algorithms for orthogonal and opposite cast removal

In this section and the next, algorithms are presented for the computation of casting planes, and hence, determine whether a given polyhedron is castable. This section focuses on orthogonal and opposite cast removal.

4.1 A simple algorithm for simple polyhedra

We compute $O(n)$ candidate casting planes as follows. By Lemma 7, we need only consider the casting planes of the convex hull of P. We first compute the candidate casting planes that intersect some edge properly, and then the ones that are generated. We only consider opposite cast removal; the case of orthogonal cast removal only requires some straightforward changes.

Let $E_1, \ldots, E_k$ be a partitioning of E into subsets of parallel edges. Let $V_i^{\subset}$ be the set of vertices that must be contained in the casting plane for the cast removal direction parallel to the edges of E_i. For each E_i, we compute the convex hull G_i^+ of the lower endpoints of E_i and the convex hull G_i^- of the upper endpoints of E_i. The convex hull of $|E_i|$ points in 3-dimensional space can be computed in $O(|E_i| \log |E_i|)$ time, see e.g. [11, 19]. Then we compute all planes that contain the vertices of $V_i^{\subset}$ and separate the interiors of G_i^+ and G_i^-, and contain at least three vertices of $V_i^{\subset}$, G_i^+ and G_i^-. This gives $O(|E_i|)$ candidate casting planes. Summed over all subsets $E_1, \ldots, E_k$, we obtain $O(n)$ candidate casting planes in $O(n \log n)$ time.

Second, we compute the other candidate casting planes, which are the generated planes of the hierarchical decomposition of Subsection 3.1. Antipodal pairs computations take $O(n)$ time and convex hull computations take $O(n \log n)$ time, see e.g. [11, 19]. The total time taken by the process is given by the recurrence $T(n) \leq T((1-\alpha)n) + O(n \log n)$, where α is the constant in the $\Omega(n)$ of Lemma 10. This recurrence solves to $T(n) = O(n \log n)$. We conclude:

Lemma 11 *Given a polyhedron P with n vertices, one can compute in $O(n \log n)$ time a set Q of $O(n)$ planes such that any casting plane h that contains at least three vertices of P is contained in Q, assuming opposite cast removal.*

Theorem 4 *Given a polyhedron P with n vertices, one can decide in $O(n^2)$ time and linear space whether P is castable when the cast parts must be removed in orthogonal or opposite directions.*

4.2 Walking around convex polyhedra

For convex polyhedra, the result can be improved. Recall that determining whether a plane h is a casting plane for P can be done by only considering the facets intersected by h and the facets incident to the edges that are contained in h (this only holds for convex polyhedra), see Section 2.2. A linear program on this set of facets tells us whether h is a casting plane. We also know, by Theorem 2, that the total number of facets that we check, for all $O(n)$ candidate casting planes, is only $O(n \log n)$. To test each candidate casting plane, either generated or a member of an equivalence class of weakly equivalent planes, our algorithm walks around the intersection of the plane with the polyhedron at a cost of $O(\log n)$ per step. This leads to an $O(n \log^2 n)$ time algorithm for a convex polyhedron P with n vertices.

Theorem 5 *Given a convex polyhedron P with n vertices, one can decide in $O(n \log^2 n)$ time and linear space whether P is castable when the cast parts must be removed in orthogonal or opposite directions.*

4.3 A data structuring approach

The second, $O(n \log^2 n)$ time solution described above only applies to convex polyhedra. By using data structures, we will improve upon the quadratic time results of Theorem 4 for simple polyhedra as well. Unfortunately, the storage requirements increase with the data structuring method. The idea is to test every candidate casting plane by querying with it in a data structure (instead of applying linear programming). The query determines whether h really is a valid casting plane for the polyhedron P. The preprocessing of the data structure should be less than quadratic and the query time should be less than linear in order to beat the quadratic time bound. It turns out that the data structure is exactly the same for the three versions of removing the cast; only the query algorithms are different.

Let P be a polyhedron. For any vertex $v \in V$, define $F(v)$ as the set of facets incident to v, and for any subset $V' \subseteq V$, define $F(V')$ as the set of facets incident to at least one vertex of V'. We make the following observation:

Observation 2 *For any plane h, we have $F_h^+ \cup F_h^\times = F(V_h^+)$.*

We store P in a 2-level data structure T. The primary tree is a 3-dimensional partition tree that stores the set V of vertices of P, see Matoušek [15, 16] and Agarwal and Sharir [2] for example. They show that for any constant $\epsilon > 0$, a structure of size and preprocessing time $O(n^{3/2+\epsilon})$ exists, such that for any query plane h, the subset $V_h^+ \subseteq V$ of vertices above h can be retrieved in $O(n^{1/2+\epsilon})$ canonical nodes in $O(n^{1/2+\epsilon})$ time. For any node δ of T, corresponding to a canonical subset V_δ, the secondary structure at δ stores V_δ as follows. Recall from Subsection 2.1 that for a facet f, $\Psi(f)$ is one of the closed half-spaces of which the bounding plane supports f, and $\Psi_0(f)$ is $\Psi(f)$ translated such that the bounding plane contains the origin. Let $F(V_\delta)$ be the set of facets incident to at least one vertex of V_δ. Let B_δ be the cone $\bigcap\{\Psi_0(f) \mid f \in F(V_\delta)\}$ with apex at the origin. The secondary structure is simply an array or balanced binary tree that stores the facets of B_δ in cyclic order around the apex. The secondary structure allows for queries with a half-line starting at the origin, to determine whether the half-line is contained in B_δ. This query is in fact a 2-dimensional query to determine whether a point lies in a convex polygon.

Suppose that we wish to determine whether h is a valid casting plane for orthogonal cast removal. Then we search with h in T and determine the canonical nodes of T with respect to h, in particular, the set $\Delta^+ = \{\delta_1, \ldots, \delta_t\}$ of nodes of which the associated sets $V_{\delta_1}, \ldots, V_{\delta_t}$ are a partition of V_h^+. Let ℓ_h^+ be the upward half-line normal to h starting at the origin. For each of the nodes δ_i, we query with ℓ_h^+ in the secondary structure to determine by binary search whether $\ell_h^+ \in B_{\delta_i}$. If the answer is positive for all nodes $\delta_1, \ldots, \delta_t$, then $P \cap h^+$ is a terrain with respect to the direction normal to h (and parallel to ℓ_h^+). The query is repeated for h^-, to determine whether $P \cap h^-$ is a terrain with respect to the direction normal to h. If this is also the case, then h is a casting plane of P for orthogonal cast removal. Since $t = O(n^{1/2+\epsilon})$, the query time is $O(n^{1/2+\epsilon} \log n)$.

Next we consider cast removal in opposite directions. The query is a variation to the previous solution. We determine both sets Δ^+ and Δ^- of canonical nodes for the queries with h^+ and h^-, respectively. Let $\Delta^+ = \{\delta_1, \ldots, \delta_t\}$ and $\Delta^- = \{\delta'_1, \ldots, \delta'_s\}$. We wish to determine whether the common intersection of the t cones $B_{\delta_1}, \ldots, B_{\delta_t}$, intersected with the reflection in the origin of the common intersection of the s cones $B_{\delta'_1}, \ldots, B_{\delta'_s}$, is non-trivial. One can decide whether the common intersection is non-trivial using the algorithm of Reichling [22]. He shows how to find an extremal point in the common intersection of k convex n-gons in $O(k \log^2 n)$ time. In our case, we 'reflect' all operations on the second set of cones. (Alternatively, we could store the reflected cones explicitly at every node and work with these, but this is not necessary.) The query time is $O((s+t)\log^2 n) = O(n^{1/2+\epsilon} \log^2 n)$ time.

Thirdly, we consider the query in the same structure to solve arbitrary cast removal. We remark that for arbitrary cast removal, we can determine using $O(n \log n)$ queries whether a casting plane exists, even though we do not have a subquadratic bound on the number of casting planes in this case. This will be shown in the next section.

Let h be the casting plane. We determine the set Δ^+ as before; let $B_{\delta_1}, \ldots, B_{\delta_t}$ be the cones that are stored at the set Δ^+ canonical nodes. Now we have to determine whether the common intersection of these cones is non-trivial. Any half-line starting at the origin and in the common intersection of the cones represents a direction with respect to which $P \cap h^+$ is a terrain. A half-line in the common intersection of the t cones can be determined in $O(t \log^2 n) = O(n^{1/2+\epsilon} \log^2 n)$ time using Reichling's algorithm. If no such half-line exists, then $P \cap h^+$ is not a terrain with respect to any direction. If $P \cap h^+$ is a terrain for some direction, we repeat the query to test whether $P \cap h^-$ is a terrain.

Theorem 6 *For any constant $\epsilon > 0$ and any simple polyhedron P with n vertices, a data structure of size and prepreocessing time $O(n^{3/2+\epsilon})$ exists, such that for any query plane h, one can determine in $O(n^{1/2+\epsilon})$ time whether h is a casting plane for P in any of the versions for removing the cast.*

Remark: In fact, we have also shown that for any query half-space h^+, we can determine within the same bounds whether $P \cap h^+$ is a terrain in some direction. Furthermore, by choosing a different partition tree for the primary tree, we can trade query time for storage space, see e.g. Chazelle, Sharir and Welzl [7], Matoušek [16] and Agarwal and Sharir [2]. For any $n \leq N \leq n^3$, a data structure of size and preprocessing time $O(N^{1+\epsilon})$ exists with query time $O(n^{1+\epsilon}/N^{1/3})$. The theorem above states the version we need for the casting problem.

Corollary 3 *For any constant $\epsilon > 0$ and any simple polyhedron P with n vertices, one can determine in $O(n^{3/2+\epsilon})$ time and space whether P is castable when the cast parts must be removed orthogonal to the casting plane, or in opposite directions.*

5 Algorithms for arbitrary cast removal

In this section we study the most general casting problem of this paper: determine whether a simple polyhedron P is castable when the cast parts may be removed in arbitrary directions. Using results of the previous sections, we can obtain an $O(n^{9/4+\epsilon})$ time algorithm: we devise a data structure as in the previous section to test all candidate $O(n^2)$ casting planes. We will do better in this section. Using Lemmas 5 and 6 and one more observation on arbitrary cast removal, we first obtain a simple $O(n^2 \log n)$ time and linear space algorithm, and then a more complicated $O(n^{3/2+\epsilon})$ time and space algorithm.

Let P be a polyhedron. We first test whether P admits opposite cast removal using the simple $O(n^2)$ time algorithm of Theorem 4. If so, we are done. Otherwise, if P is convex, then by Lemma 5 we only have to consider casting planes that contain some edge of P. If P is non-convex, then, by Lemma 6, we only have to consider casting planes that contain an edge of the convex hull of P.

Observation 3 *Let P be a polyhedron and h a plane that contains an edge e of the convex hull of P. Assume w.l.o.g. that e is horizontal and that a vertical plane exists which supports e and has $P - cl(e)$ completely to the one side.*

- *If $P \cap h^+$ is a terrain and $P \cap h^-$ is not a terrain, then no plane μ containing e for which $P \cap h^- \subset \mu^-$ is a casting plane.*

- *If $P \cap h^+$ is not a terrain and $P \cap h^-$ is a terrain, then no plane μ containing e for which $P \cap h^+ \subset \mu^+$ is a casting plane.*

- *If $P \cap h^+$ and $P \cap h^-$ are both not a terrain, then no plane containing e is a casting plane.*

The above observation sets up a binary search for a casting plane that contains some edge e of the convex hull of P. A plane h rotating about e encounters the $n-2$ other vertices of P in some order. One binary search step on this order tests whether $P \cap h^+$ and $P \cap h^-$ are terrains using linear programming. Therefore, we have the following.

Theorem 7 *Given a simple polyhedron P with n vertices, one can determine in $O(n^2 \log n)$ time and linear space whether a casting plane for P exists, when the cast parts can be removed in arbitrary directions.*

As shown in Subsection 4.3, we can preprocess P into a data structure such that any plane can be tested in $O(n^{1/2+\epsilon})$ time. Since $O(n \log n)$ planes are tested by the above procedure, the test part can be improved to $O(n^{3/2+\epsilon})$ time. However, how can we obtain the order of $v_1, \ldots, v_{n-2}$ without sorting? Again, the solution lies in data structuring. Notice first that the order of $v_1, \ldots, v_{n-2}$ is not needed explicitly. In the first step, the vertex that is the median $v_{n/2-1}$ must be determined, and in the following steps a median in one of the two halves.

The data structure that is needed solves the following query problem: Given a query edge e such that there exists a plane h containing e that has the interior of P completely to the one side, and an integer k, find the k-th vertex of P that is encountered by h when it rotates about e. Dualization yields a more familiar query problem: preprocess a set of n planes (dual to the vertices of P), such that for any given query ray, the k-th intersection point with the planes can be determined. The query ray is contained in the line dual to the line supporting e, and the starting point of the ray is any point dual to a plane containing e that has the interior of P completely to the one side.

Let $\mathcal{D}(V) = \{\mathcal{D}(v_i) \,|\, v_i \in P\}$ be the set of planes dual to the vertices of P, preprocess them into a data structure for line segment intersection counting, as given by Agarwal and Matoušek [1]. They show that for any $\epsilon > 0$, a structure of size and preprocessing time $O(n^{3/2+\epsilon})$ exists, such that segment intersection counting queries can be answered in $O(n^{1/2+\epsilon})$ time. Furthermore, they show how the same structure can be used to find the k-th intersection point of a query ray with $\mathcal{D}(V)$ in $O(n^{1/2+\epsilon})$ time. Let the query ray be parameterized by $q + \lambda \cdot \vec{u}$, $\lambda \geq 0$, where q is a point and $\vec{u}$ is a vector in 3-dimensional space. In our application, if the k-th intersection point does not exist, and the last intersection point is the j-th, then the query should be continued in 'wrap-around' mode: find the $(k - j)$-th plane for the query ray along the same line and in the same direction, but with q translated in direction $-\vec{u}$ to infinity. When this happens, the plane rotating about e in primal space rotates past a vertical orientation. These adaptations to the query algorithm can easily be made within the same asymptotic time bounds. Hence, we conclude:

Theorem 8 *For any simple polyhedron P with n vertices and any constant $\epsilon > 0$, one can determine in $O(n^{3/2+\epsilon})$ time and space whether P is castable, when the cast parts can be removed in arbitrary directions.*

6 Conclusions and open problems

This paper studied the geometric version of the problem of determining whether a simple polyhedron can be manufactured using casting. It was assumed that there are two cast parts, and each has be removed with a single translation. We presented simple algorithms that use $O(n^2)$ or $O(n^2 \log n)$ time and linear space which are based on linear programming. Furthermore, we showed that theoretically, better results can be obtained using partition trees and their variants. This leads to an $O(n^{3/2+\epsilon})$ time and space algorithm. The n^{ϵ}-factors can be replaced by polylogarithmic factors if we use the partition trees of Matoušek [16]. Finally, as a by-product, we obtained a combinatorial bound on the number of planes intersecting a polyhedron in edges only, and on the summed number of edges in these planes.

Manufacturing applications have not been studied much in computational geometry. There are quite a large number of open problems to be solved in this area. We first list some open problems related to this paper, and then list a few others in cast design that deserve attention.

1. What is the maximum number of distinct casting planes in case of arbitrary cast removal? This paper shows $O(n^2)$, whereas the only lower bound we have is linear.

2. For a convex polyhedron P, what is the maximum summed complexity of the intersection of all weakly distinct casting planes with P? This paper shows $O(n \log n)$ in case of opposite cast removal, but the trivial lower bound is linear.

3. Give simple algorithms for casting that improve our simple $O(n \log^2 n)$, $O(n^2)$ and $O(n^2 \log n)$ time algorithms. Find algorithms that improve upon our $O(n^{3/2+\epsilon})$ time algorithms. (De Berg [9] observed that orthogonal cast removal for simple polyhedra can be done in $O(n^{4/3+\epsilon})$ time by using a different data structure to test candidate casting planes.)

4. Suppose that we wish to determine castability of an object with non-linear boundaries. Give (simple) algorithms that deal with this problem.

5. For some casting processes, is not necessary that the cast parts must be separated by a plane. In these cases, every convex polyhedron is castable. However, no algorithms are known for cast removal of simple polyhedra.

Acknowledgements. This research was initiated during the workshop on 'Injection molding' at the Bellairs Research Institute, McGill University. The authors thank the organiser Godfried Toussaint and the other participants for creating a good research environment. Furthermore, we thank Pankaj Agarwal for answering several of our questions.

References

[1] Agarwal, P. K., and J. Matoušek, Ray shooting and parametric search. *Proc. 24th ACM Symp. Theory of Computing* (1992), pp. 517–526.

[2] Agarwal, P. K., and M. Sharir, Applications of a new space partitioning technique. *Proc. 2nd WADS* (1991), Lect. Notes in Comp. Science 519, Springer-Verlag, pp. 379–391.

[3] Asberg, B., G. Blanco, P. Bose, J. Garcia, M. Overmars, G. Toussaint, G. Wilfong, and B. Zhu, Feasibility of design in stereolithography. *Proc. 13th Symp. on FST & TCS* (1993), Lect. Notes in Comp. Science 761, pp. 228–237.

[4] Bose, P., D. Bremner, and M. van Kreveld, Determining the Castability of Simple Polyhedra. Tech. Rep. SOCS 93.12, School of Comp. Science, McGill University, 1993.

[5] Bose, P., M. van Kreveld, and G. Toussaint, Filling polyhedral molds. *Proc. 3rd WADS* (1993), Lect. Notes in Comp. Science 709, Springer-Verlag, pp. 210–221.

[6] Bose, P., and G. Toussaint, Geometric and computational aspects of injection molding. *Proc. 3rd Int. Conf. on CAD and Computer Graphics* (1993), Beijing, China, pp. 237–242. Also: Tech. Rep. SOCS 92.16, School of Comp. Science, McGill University, 1992.

[7] Chazelle, B., M. Sharir, and E. Welzl, Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Proc. 6th ACM Symp. on Comp. Geometry* (1990), pp. 23–33.

[8] Personal Communication with R. Connolly, Manager Injection Molding, Industrial Materials Institute, National Research Council of Canada, Boucherville, Quebec, November 1993.

[9] De Berg, M., personal communication, 1994.

[10] Dobkin, D. P., and D. G. Kirkpatrick, A linear time algorithm for determining the separation of convex polyhedra. *J. of Algorithms* **6** (1985), pp. 381–392.

[11] Edelsbrunner, H., *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, 1987.

[12] Elliott, R., *Cast iron technology.* Butterworths, London, 1988.

[13] Fekete, S.P., and J.S.B. Mitchell, *Geometric aspects of injection molding,* manuscript, 1993.

[14] Held, M., *On the Computational Geometry of Pocket Machining.* Lect. Notes in Comp. Science 500, Springer-Verlag, 1991.

[15] Matoušek, J., Efficient partition trees. *Proc. 7th ACM Symp. on Comp. Geometry* (1990), pp. 1–9.

[16] Matoušek, J., Range searching with efficient hierarchical cuttings. *Proc. 8th ACM Symp. on Comp. Geometry* (1991), pp. 276–285.

[17] McAllister, M., and J. Snoeyink, Two dimensional Computation of the Three dimensional Reachable Region for a Welding Head. *Proc. 5th Canadian Conf. on Comp. Geom* (1993), pp. 437-442.

[18] Megiddo, N., Linear-time algorithms for linear programming in R^3 and related problems. *SIAM J. Comp.* **12** (1983), pp. 759–776.

[19] Preparata, F.P., and M.I. Shamos, *Computational geometry – an introduction.* Springer-Verlag, 1985.

[20] Pribble, W.I., Molds for reaction injection, structural foam and expandable styrene molding. In: *Plastics mold engineering handbook*, J.H. DuBois and W.I. Pribble (Eds.), Van Nostrand Reinhold Company Inc., New York, 1987.

[21] Rappaport, D., and A. Rosenbloom, Moldable and Castable Polygons. *Proc. 4th Canadian Conf. on Comp. Geom.* (1992), pp. 322-327.

[22] Reichling, M., On the detection of a common intersection of k convex objects in the plane. *Inf. Proc. Lett.* **29** (1988), pp. 25–29.

[23] Walton, C.F., and T.J. Opar (Eds.), *Iron castings handbook.* Iron casting society, Inc., 1981.

[24] Whelan, A., *Injection moulding machines.* Elsevier, London, 1984.

A Fast Algorithm For Constructing Sparse Euclidean Spanners

(Extended Abstract)

Gautam Das [*][†] Giri Narasimhan[†]

Abstract

Let $G = (V, E)$ be a n-vertex connected graph with positive edge weights. A subgraph G' is a t-spanner if for all $u, v \in V$, the distance between u and v in the subgraph is at most t times the corresponding distance in G. We design an $O(n \log^2 n)$ time algorithm which, given a set V of n points in k-dimensional space, and any constant $t > 1$, produces a t-spanner of the complete Euclidean graph of V. This algorithm retains the spirit of a recent $O(n^3 \log n)$-time greedy algorithm which produces t-spanners with a small number of edges and a small total edge weight; we use graph clustering techniques to achieve a more efficient implementation. Our spanners have similar size and weight sparseness as those constructed by the greedy algorithm.

1 Introduction

Let $G = (V, E)$ be a n-vertex connected graph with positive edge weights. A subgraph G' is a t-spanner if for all $u, v \in V$, the distance between u and v in the subgraph is at most t times the corresponding distance in G. The value of t is known as the *stretch factor* of G'. Let V be a set of n points in k-dimensional space. An *Euclidean graph* has V as its vertices, and its edges are straight line segments joining pairs of points, with edge weights being their Euclidean distances. The *complete* Euclidean graph contains all $n(n-1)/2$ edges.

Although complete graphs represent ideal communication networks, they are expensive to build in practice, and sparse spanners represent low cost alternatives. Sparseness of spanners is usually measured by various criteria such as few edges, small weight, and small degree. Spanners for complete Euclidean graphs as well as for arbitrary weighted graphs find applications in robotics, network topology design, distributed systems, design of parallel machines, and have also been investigated by graph theorists. Most of the recent literature on spanners are referenced in [4, 9].

Let MST refer to a minimum spanning tree of the complete graph. For complete Euclidean graphs in 2-dimensions, an $O(n \log n)$-time algorithm is described in [7] which produces t-spanners with $O(n)$ edges and weight $O(wt(MST))$, which is optimal. The problem gets more difficult in higher dimensions. In k-dimensional space, there are several algorithms that run in $O(n \log n)$ time [8, 10]. However they only guarantee that the number of spanner edges is $O(n)$; the *total weight* could be arbitrarily large. In [4] an $O(n \log n)$ time algorithm is described which generates spanners in k-dimensions with weight $O(\log n) \cdot wt(MST)$. The nature of this algorithm is such that the $O(\log n)$ factor cannot be eliminated, so the algorithm always produces a suboptimally weighted spanner.

In contrast, in [1, 4, 5] a simple *greedy algorithm* is presented which generates spanners with $O(n)$ edges *and* a weight which is likely to be asymptotically optimal for all dimensions. In [4] it is shown that for any dimension k the weight is $O(\log n) \cdot wt(MST)$, and in [5] it is shown that for $k \leq 3$ the weight is $O(wt(MST))$. In [5] it is conjectured that the weight is actually asymptotically optimal in any dimension and that the $O(\log n)$ factor can be elim-

[*] Supported in part by NSF grant CCR-930-6822

[†] Math Sciences Dept., Memphis State University, Memphis, TN 38152. *e-mail:* dasg/giri@next1.msci.memst.edu

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

inated by a more careful analysis. The constants implicit in the O-notation depend only on t and k. The greedy algorithm has also been used to generate spanners of arbitrary weighted graphs, and there again, it is superior to other algorithms in the sense that it produces spanners with very small weight.

However, the greedy algorithm is handicapped by a slow running time, mainly because it has to make a large number of shortest path queries. In the Euclidean case, the best-known implementation has a running time of $O(n^3 \log n)$. In this paper, we design an $O(n \log^2 n)$-time algorithm to construct t-spanners, which retains the spirit of the greedy algorithm. This new algorithm is faster because we use *graph clustering* techniques to answer shortest path queries only approximately. The spanners produced by this algorithm have the same asymptotic size and weight bounds as the spanners produced by the greedy algorithm. Furthermore, any improvements in these bounds for the greedy algorithm spanners is likely to lead to similar improvements in the bounds for the spanners produced by this algorithm. Graph clustering is not new, and in fact there are several algorithms which use a variety of clustering techniques to generate spanners of Euclidean as well as arbitrary weighted graphs [2, 3, 6, 8, 10]. What is interesting is the way in which we exploit a simple clustering technique in our implementation.

In the next section we introduce notation, summarize previous relevant research, and give an overview of our algorithm. Section 3 describes the clustering technique used. Section 4 describes the algorithm. We conclude with some open problems.

2 Preliminaries

Let $G = (V, E)$ be a n-vertex connected graph with positive edge weights. The weight of a single edge e is $wt(e)$, the weight of a set of edges E' is $wt(E')$, and the weight of the entire graph is $wt(G)$. Let $sp_G(u, v)$ be the weight of the shortest path in G between u and v. Let $d(u, v)$ refer to the Euclidean distance between points u and v in Euclidean space.

In [1, 4], a *greedy algorithm* is developed for constructing t-spanners of arbitrary weighted graphs, for any $t > 1$. We reproduce the algorithm in Figure 1. Let us apply this algorithm to a complete Euclidean graph in k-dimensional space and let the output graph be $G' = (V, E')$.

It is easy to see that the output is a t-spanner, since for any (u, v) that is not an edge of G', $sp_{G'}(u, v) \leq t \cdot d(u, v)$. *In addition*, for any (u, v) that is an edge of G', the weight of the second shortest path between u and v in G' is greater than $t \cdot d(u, v)$. The second property is crucial in showing that the total weight of the spanner is small. In [4] it is shown that G' has $O(n)$ edges and has a total edge weight of $O(\log n) \cdot wt(MST)$, where MST is a minimum spanning tree of V. In [5] it is shown that for $k \leq 3$, the greedy algorithm outputs a t-spanner of weight $O(wt(MST))$. In that paper it is conjectured that this result is true for k-dimensions.

```
Algorithm GREEDY($G = (V, E), t$)
begin
order $E$ by non-decreasing weights
$E' \leftarrow \emptyset$, $G' \leftarrow (V, E')$
for each edge $e = (u, v) \in E'$ do
        if $sp_{G'}(u, v) > t \cdot wt(e)$ then
                $E' \leftarrow E' \cup \{e\}$, $G' \leftarrow (V, E')$
output $G'$
end.
```

Figure 1: The greedy algorithm

In this paper, our algorithm mimics the greedy algorithm and constructs a t-spanner G' with asymptotically same size and weight sparseness. Apart from the very short edges of the spanner, all other edges (u, v) satisfy the property that the weight of the second shortest path between u and v in G' is greater than $c \cdot d(u, v)$, where c is a constant dependent on t. Thus, techniques similar to those used in [4, 5] can be used to prove the same sparseness bounds. Details appear later.

The following concept will be useful in later sections. We define a certain kind of "partial" spanners of complete Euclidean graphs. Let V be a set of n points in k-dimensional space, $t > 1$ and $W > 0$. A graph G' is a (t, W)-spanner if for all $u, v \in V$ such that $d(u, v) \leq W$, $sp_{G'}(u, v) \leq t \cdot d(u, v)$. In other words, G' provides short paths between pairs of points whose inter-distance is at most W. For instance, at any stage of the greedy algorithm, if the edge most recently examined has weight L, the partially constructed spanner is a (t, L)-spanner.

2.1 Overview of the algorithm

We first run the $O(n \log n)$ time algorithm in [8] to get a graph $G = (V, E)$, which is a $\sqrt{t}$-spanner of the complete Euclidean graph. Although it may have a large weight, it has $O(n)$ edges. We shall then find a

$\sqrt{t}$-spanner G' of G. Clearly, G' will be a t-spanner of the complete graph.

If we were to run the original greedy algorithm on G to get G', it will have to answer $O(n)$ shortest path queries on G'. Our algorithm does better by speeding up the queries. We group the edges in E into $O(\log n)$ groups, such that each group has edges that are more or less of equal weight. Then we examine the edges, going from one group to the next in order of increasing weight, and at the same time building a partially constructed spanner, G'.

Immediately before processing a group of edges (of weight approximately W), the algorithm creates a *cluster-graph* H as follows. Roughly speaking, the vertices of G' are covered by a set of *clusters* such that each cluster consists of vertices that are within $O(W)$ distance from each other in G'. Each cluster is collapsed into a representative vertex, namely its *center*, to form a single macro vertex of H. These macro vertices are connected by edges of weight $\Theta(W)$ only if the distance between their centers in G' is $\Theta(W)$. Notice that all edge weights of H are more or less the same. Thus distances in G' are approximated by distances in H. A crucial observation, as proved later, is that in the Euclidean case the maximum degree of H turns out to be a constant. Thus the shortest path query can be answered in *constant time* as follows. Start with a macro vertex whose corresponding cluster contains u, and see whether v belongs to any of the clusters that are within $O(\sqrt{t})$ links from the start cluster. Since the degree is constant, and t is a constant, we need only explore a constant-sized portion of H. Once we have examined all the edges in a group, we create a cluster-graph again (but with a larger W), and process the next group of edges.

The running time of of our algorithm is dominated by the periodic clustering necessary.

3 Clustering a graph

In this section we first consider arbitrary weighted graphs. Let $G' = (V, E')$ be an any positive weighted graph, and $W > 0$. A *cluster* C is a set of vertices, with one vertex v its *center*, such that for all $u \in V$, $sp_{G'}(u,v) \leq W$ implies that $u \in C$. W is the *radius* of the cluster. A *cluster-cover* is a set of clusters $C_1, C_2, \ldots, C_m$, (with corresponding centers $v_1, v_2, \ldots, v_m$) such that their union is V. Clusters in a cluster-cover may overlap, however a cluster center does not belong to any other cluster.

We describe an obvious algorithm to compute a cluster-cover. First consider a procedure called SINGLE-SOURCE(G', v, W), which takes as input any weighted graph G', a start vertex v, and $W > 0$. It outputs all vertices u such that $sp_{G'}(v,u) \leq W$. It can be implemented as Dijkstra's single-source shortest path algorithm by using a heap and ensuring that the algorithm never visits vertices further than W from v. We now design a procedure CLUSTER-COVER(G', W) as follows. Select any vertex v_1 of G', and run SINGLE-SOURCE(G', v_1, W). The output is a cluster C_1 with center v_1. Then select any vertex v_2 not yet visited, and run SINGLE-SOURCE(G', v_2, W). The output is C_2 with center v_2. Continue this process until all vertices are visited. For each vertex, the procedure also computes the clusters to which it belongs as well as its distance from their respective centers.

Let $G' = (V, E')$ be any weighted graph, $W > 0$ and $0 < \delta < 1/2$. Assume a cluster-cover has been constructed with cluster radius δW. A *cluster-graph* H is defined as follows. The vertex set is V. The are two kinds of edges, *intra-cluster edges* and *inter-cluster edges*. For all C_i, and for all $u \in C_i$, $[u, v_i]$ is an intra-cluster edge (we use square brackets to distinguish cluster-graph edges from the edges of G'). Inter-cluster edges are between cluster centers. $[v_i, v_j]$ is an inter-cluster edge if, either $sp_{G'}(v_i, v_j) \leq W$, or there exists $e = (u,v) \in E'$ such that $u \in C_i, v \in C_j$. Cluster-graph edges have weights, and $wt([u,v])$ is defined as $sp_{G'}(u,v)$.

We now describe a procedure called CLUSTER-GRAPH(G', δ, W). It first calls CLUSTER-COVER($G', \delta W$). In fact, during this process all intra-cluster edges (and their weights) will also be computed. Recall that inter-cluster edges have to satisfy one of two conditions. Inter-cluster edges satisfying the first condition may be computed by running SINGLE-SOURCE on G' from each cluster center v_i and checking which other cluster centers are no further than W from v_i. The remaining inter-cluster edges may be computed as follows. For all $e = (u,v) \in E'$ do the following. For all cluster centers v_i, v_j such that u belongs to v_i's cluster and v belongs to v_j's cluster, add the inter-cluster edge $[v_i, v_j]$ with weight defined to be $wt([v_i, u]) + wt(e) + wt([v, v_j])$ (unless the inter-cluster edge already existed with a smaller weight).

The motivation for the cluster-graph is that it is a simpler structure than G'. Shortest path queries in G' can be translated into shortest path queries in H, and the latter can be performed more efficiently. The following two lemmas are obvious.

Lemma 3.1 *Intra-cluster edges are no larger that δW. Inter-cluster edges are larger than δW, but no larger than* $\max\{W, D + 2\delta W\}$, *where D is the weight of the largest edge in G'.*

Lemma 3.2 *Let a path between u and v in H have weight L. Then there is a path between u and v in G' with weight at most L.*

The next lemma is the converse, which is harder. We first introduce some definitions. A vertex u is defined to be ***sufficiently far*** from vertex v if, (1) no single cluster contains both, and (2) $sp_{G'}(u, v) > W - 2\delta W$. Define a ***cluster-path*** in H to be a path where the first and last edges are intra-cluster edges, but all intermediate edges are inter-cluster edges.

Lemma 3.3 *Let u be sufficiently far from v. Let L_1 be the weight of a path between u and v in G'. Then there exists a cluster-path between u and v in H with weight L_2 such that*

$$L_2 \leq \left(\frac{1+6\delta}{1-2\delta}\right) \cdot L_1$$

Proof : This lemma says that H approximates paths in G' only for pairs of vertices which are sufficiently far apart. Secondly, notice that the quality of the approximation depends on δ; smaller values of δ make paths in H closer in weight to paths in G', although H becomes denser.

Let the path from u to v having weight L_1 in G' be P. We shall use the notation $P(w, x)$ to denote the vertices of P between vertices w and x, not including w. We shall construct a cluster-path Q from u to w. in H with weight L_2 as follows. follows. Let C_0 be any cluster (with center v_0) containing u. The first edge of Q is the intra-cluster edge $[u, v_0]$. Next, among all clusters with centers adjacent to v_0 in H, let C_1 (with center v_1) intersect the furthest vertex along $P(u, v)$, say w_1. Add the inter-cluster edge $[v_0, v_1]$ to Q. Next, among all clusters with centers adjacent to v_1 in H, let C_2 (with center v_2) intersect the furthest vertex along $P(w_1, v)$, say w_2. Add the inter-cluster edge $[v_1, v_2]$ to Q. This process may be continued until we reach a cluster center, v_m, whose cluster contains v. At this stage, complete Q by adding the intra-cluster edge $[v_m, v]$. Figure 2 illustrates some of this notions.

Case 1: $m = 1$. In this case there is only one inter-cluster edge along Q. Since u is sufficiently far from v, we know that

$$L_1 > W - 2\delta W$$

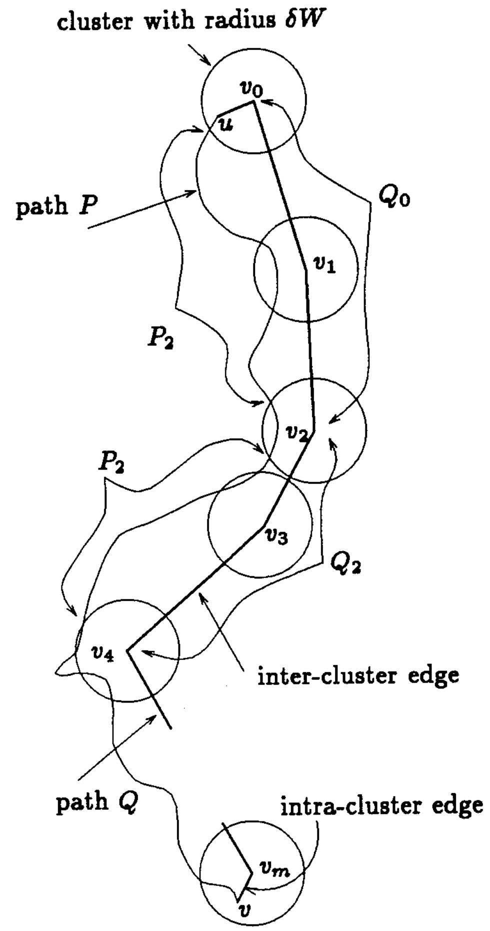


Figure 2: Paths in G' and H

Now $L_2 = wt([u, v_0]) + wt([v_0, v_1]) + wt([v_1, v])$. But $wt([u, v_0])$ and $wt([v_1, v])$ are each at most δW, while $wt([v_0, v_1]) = sp_{G'}(v_0, v_1) \leq sp_{G'}(v_0, u) + sp_{G'}(u, v) + sp_{G'}(v, v_1) \leq 2\delta W + L_1$. Thus

$$L_2 \leq L_1 + 4\delta W$$

We now have two inequalities relating L_1, L_2 and W. After eliminating W, we get

$$L_2 < \left(\frac{1+2\delta}{1-2\delta}\right) \cdot L_1$$

Case 2: $m \geq 2$ and is even. In this case, suppose $[v_i, v_{i+1}]$ and $[v_{i+1}, v_{i+2}]$ are any two consecutive inter-cluster edges on Q. We observe that the sum of their weights is greater than W. If this were not so, then the edge $[v_i, v_{i+2}]$ (which exists by the definition of the cluster-graph) would have instead been added to Q by the above process.

We divide Q into portions $Q_0, Q_2, \ldots$, where Q_{2i} is the portion between v_{2i} and v_{2i+2}. Similarly, we divide P into portions $P_0, P_2, \ldots$, where P_{2i} is the portion between the last vertex intersecting C_{2i} and the first vertex intersecting C_{2i+2} (see Figure 2). We shall first prove that for any even i, the weight of Q_{2i} is no more than a constant (which depends upon δ) times the weight of P_{2i}.

Let the weight of P_{2i} be p_{2i} and that of Q_{2i} be q_{2i}. Since there cannot be an inter-cluster edge between v_{2i} and v_{2i+2}, we have

$$p_{2i} > W - 2\delta W$$

Select r to be any vertex of P_{2i} within the intermediate cluster C_{2i+1}. The vertex r splits P_{2i} into two portions. Let p'_{2i} (p''_{2i}) be the weight of the initial (final) portions; thus $p_{2i} = p'_{2i} + p''_{2i}$. So $wt([v_{2i}, v_{2i+1}]) \leq p'_{2i} + 2\delta W$, and similarly $wt([v_{2i+1}, v_{2i+2}]) \leq p''_{2i} + 2\delta W$. Adding the two, we get

$$q_{2i} \leq p_{2i} + 4\delta W$$

We now have two inequalities relating p_{2i}, q_{2i} and W. After eliminating W, we get

$$q_{2i} < \left(\frac{1+2\delta}{1-2\delta}\right) \cdot p_{2i}$$

Summing over all even values of i, and taking into account the two intra-cluster edges at either ends of Q, we get

$$L_2 < \left(\frac{1+2\delta}{1-2\delta}\right) \cdot L_1 + 2\delta W$$

Since u is sufficiently far from v, we know that $L_1 > W - 2\delta W$. That is, $L_1 \cdot 2\delta/(1-2\delta) > 2\delta W$. Substituting for $2\delta W$ in the above inequality and simplifying, we get

$$L_2 < \left(\frac{1+4\delta}{1-2\delta}\right) \cdot L_1$$

Case 3: $m \geq 3$ and is odd. In this case, the analysis will be exactly the same as in Case 2, except that we have to account for the last inter-cluster edge along Q and correspondingly the portion of P between the last two clusters. Let q_{m-1} be the weight of $[v_{m-1}, v_m]$, and let p_{m-1} be the weight of the portion of P between the last vertex intersecting C_{m-1} and first vertex intersecting C_m. Clearly $q_{m-1} \leq p_{m-1} + 2\delta W$. The inequality does not change if we rewrite it as $q_{m-1} \leq \left(\frac{1+2\delta}{1-2\delta}\right) \cdot p_{m-1} + 2\delta W$. We then sum up the p's and q's as in Case 2, and get

$$L_2 < \left(\frac{1+2\delta}{1-2\delta}\right) \cdot L_1 + 4\delta W$$

Since $L_1 > W - 2\delta W$, we get $L_1 \cdot 4\delta/(1-2\delta) > 4\delta W$. Substituting for $4\delta W$ in the above inequality and simplifying, we get

$$L_2 < \left(\frac{1+6\delta}{1-2\delta}\right) \cdot L_1$$

The constant in Case 3 dominates, which proves the lemma. □

3.1 Clustering of partial Euclidean spanners

Although described for arbitrary weighted graphs, in our algorithm we will always be computing cluster-graphs for partial spanners of complete Euclidean graphs under the following conditions. Let $t > 1$, $\alpha > 0$, $0 < \delta < 1/2$, $\theta > 0$, and $W > 0$. For a set V of n points in k-dimensional space, let G' be a $(t, \alpha W)$-spanner such that all its edges are smaller than θW. Run CLUSTER-GRAPH(G', δ, W), and let H be the computed cluster-graph. H has several additional properties, which are used in analyzing the running time of our algorithm.

Lemma 3.4 *Let $\beta > 0$ be any constant. Inside any sphere of radius βW there can be at most a constant (which depends upon α, β, δ, t, and k) number of cluster centers.*

Proof : We first provide a lower bound on the spatial separation between any pair of cluster centers. Let v_i, v_j be two cluster centers, and let $d(v_i, v_j) = L$. There are two cases, either $L > \alpha W$ or $L \leq \alpha W$. If it is the latter case, then $sp_{G'}(v_i, v_j) \leq tL$. But $sp_{G'}(v_i, v_j) > \delta W$ (Lemma 3.1), hence $L > \delta W/t$. So in either case, $L > \min\{\alpha W, \delta W/t\}$.

Given a sphere of radius βW, we can pack at most a constant (which depends upon α, β, δ, t, and k) number of points inside it, such that the spatial separation between any pair of points is greater than $\min\{\alpha W, \delta W/t\}$. □

Lemma 3.5 *A vertex belongs to at most a constant (which depends upon α, δ, t, and k) number of clusters.*

Proof : Consider any vertex u. Let C be a cluster containing u, with cluster center v. $d(u,v) \leq sp_{G'}(u,v) \leq \delta W$. Consider a sphere of radius δW centered at u. The cluster centers of all clusters containing u have to lie within the sphere. Invoking Lemma 3.4 completes the proof. □

Lemma 3.6 *The subgraph of H induced by the inter-cluster edges has at most a constant degree (which depends α, δ, θ, t, and k).*

Proof : Let $[v_i, v_j]$ be an inter-cluster edge. By Lemma 3.1, $d(v_i,v_j) \leq sp_{G'}(v_i,v_j) \leq \max\{W, \theta W + 2\delta W\}$. Consider a sphere of radius $\max\{W, \theta W + 2\delta W\}$ centered at v_i. The cluster centers adjacent to v_i have to lie within the sphere. Invoking Lemma 3.4 completes the proof. □

We introduce one more procedure to be used by the algorithm. Let G' be a $(t, \alpha W)$-spanner satisfying the conditions laid out earlier, and let H be its cluster-graph. The procedure CHECK-PATH(H, u, v, L) returns "true" if there is a cluster-path from u to v in H with weight at most L, and "false" otherwise.

This procedure is always called by our spanner algorithm under fortunate circumstances. There is a constant $\gamma > 0$ such that, the input values of L are always at most γW. Because of this restriction, the procedure can be made to run in *constant time.* To see this, consider initiating from u a brute-force search for such a cluster-path. There are at most a constant number of intra-cluster edges leading from u to different cluster centers. The inter-cluster edges have each weight at least δW, thus if such a cluster-path exists, it can have at most γ/δ inter-cluster edges. We also know that the subgraph of H induced by inter-cluster edges has a constant degree (Lemma 3.6). We can conclude that the brute-force search initiated from u will only have to examine a *constant sized* subgraph of H, and verify whether v belongs to this subgraph, and if so, whether there exists a cluster-path from u to v of weight at most L.

This procedure is used in lieu of the time consuming shortest path queries of the original greedy algorithm.

4 The spanner algorithm

In this section we illustrate an $O(n \log^2 n)$ time algorithm for constructing t-spanners of complete Euclidean graphs in k-dimensional space. The algorithm is shown in Figure 3, and one of its subroutines is shown in Figure 4. As one can see, it retains the spirit of the greedy algorithm, except for the graph clusterings.

```
Algorithm FAST-GREEDY$(V,t)$
begin
$\delta \leftarrow \frac{1}{2}\left(\frac{\sqrt[4]{t}-1}{\sqrt[4]{t}+3}\right)$
use algorithm in [8], create a $\sqrt{t}$-spanner $G=(V,E)$
order $E$ by non-decreasing weight
let the largest edge in $E$ have weight $D$
create the intervals
        $I_0 = (0, D/n]$,
        $I_i = (2^{(i-1)}D/n, 2^i D/n]$ for $i = 1, 2, \ldots, \log n$
let $E_i$ be the (sorted) edges of $E$ with weights in $I_i$
$E' \leftarrow E_0$, $G' \leftarrow (V, E')$
for $i \leftarrow 1$ to $\log n$ do
        $W_i \leftarrow 2^{(i-1)}D/n$
        $H \leftarrow$ CLUSTER-GRAPH$(G', \delta, W_i)$
        for each edge $e \in E_i$ do
                EXAMINE-EDGE$(e)$
output $G'$
end.
```

Figure 3: An $O(n \log^2 n)$-time algorithm

4.1 Analysis of the algorithm

We analyze three aspects of the FAST-GREEDY algorithm. First, we prove that G' is indeed t-spanner. We then show that the spanner has a small weight. Finally we show that a proper implementation runs in $O(n \log^2 n)$ time.

Lemma 4.1 *The graph G' produced by the* FAST-GREEDY *algorithm is a t-spanner of the complete Euclidean graph.*

Proof : To see that G' is a t-spanner, consider any edge $e = (u,v)$ of E which is not added to E' in procedure EXAMINE-EDGE. A cluster-path (in fact, any path) from u to v in the cluster-graph corresponds to an equivalent path in G' with the same weight (Lemma 3.2). Thus e is discarded if $sp_{G'}(u,v) \leq \sqrt{t} \cdot d(u,v)$, which implies that G' is a $\sqrt{t}$-spanner of G, and is thus a t-spanner of the complete graph. □

Let us now estimate the weight of the spanner. The E_0 edges do not contribute much because their

```
Algorithm EXAMINE-EDGE($e = (u, v)$)
begin
if not CHECK-PATH($H, u, v, \sqrt{t} \cdot d(u, v)$) then
        $E' \leftarrow E' \cup \{e\}$, $G' \leftarrow (V, E')$
        for all cluster centers $w, x$ such that
                    $u$ is in $w$'s cluster and
                    $v$ is in $x$'s cluster
        do
                add inter-cluster edge $[w, x]$ to $H$
                $wt([w, x]) \leftarrow wt([w, u]) + d(u, v) +$
                    $wt([v, x])$
end.
```

Figure 4: Deciding whether an edge should be added

total weight is at most D, and $D \le wt(MST)$. We shall estimate $wt(E' \setminus E_0)$.

Lemma 4.2 *Let $e = (u, v) \in E' \setminus E_0$. Then the weight of the second shortest path between u and v in G' is greater than $\sqrt[4]{t} \cdot d(u, v)$.*

Proof : Let C be the shortest simple cycle in G' containing e. We have to estimate $wt(C) - d(u, v)$. Let $e_1 = (u_1, v_1)$ be the largest edge on the cycle. Then $e_1 \in E \setminus E_0$, and among the cycle edges it is examined last by the algorithm. Let us consider the scenario while the algorithm is examining e_1.

There is an alternate path in G' from u_1 to v_1 of weight $wt(C) - d(u_1, v_1)$. But since the algorithm eventually decides to add e_1 to the spanner, at that moment the weight of each cluster-path from u_1 to v_1 is larger than $\sqrt{t} \cdot d(u_1, v_1)$. Notice that $wt(u_1, v_1)$ is larger than W_i, where δW_i is the current cluster radius. This implies that u_1 and v_1 are not contained in any one cluster. Thus u_1 is sufficiently far from v_1. By Lemma 3.3 this implies that the weight of each path in G' from u_1 to v_1, in particular $wt(C) - d(u_1, v_1)$, is larger than $\sqrt{t} \cdot \left(\frac{1-2\delta}{1+6\delta}\right) \cdot d(u_1, v_1)$. Substituting the value of δ selected by the algorithm, this simplifies to $\sqrt[4]{t} \cdot d(u_1, v_1)$. Since $wt(C) - d(u, v) \ge wt(C) - d(u_1, v_1)$, the lemma is proved. □

Lemmas 4.1 and 4.2 are crucial in proving that $E' \setminus E_0$ has a small weight. Using techniques almost identical to those in [4, 5], we can derive a bound on the weight of $E' \setminus E_0$, which is asymptotically the same as the weight of the spanner produced by the greedy algorithm. We omit details from this version of the paper.

We now analyze the running time of the algorithm.

Lemma 4.3 *At any stage, if the most recently examined edge has weight L, at that stage G' is a $(t, L/t)$-spanner.*

Proof : Consider in general any t-spanner G' and let u and v be any pair of vertices. Then $sp_{G'}(u, v) \le t \cdot d(u, v)$, which implies that any edge on this path has weight at most $t \cdot d(u, v)$. Thus in our case, the edges to be examined in the future (weights larger than L) cannot contribute to short spanner paths between points whose spatial separation is at most L/t. □

The above lemma implies that each cluster-graph H generated by the CLUSTER-GRAPH procedure (prior to the examination of edges in E_i) satisfies all the properties and lemmas discussed in Section 3.1. During the processing of E_i, new inter-cluster edges may be added by the EXAMINE-EDGE procedure, but H still satisfies those properties.

Lemma 4.4 FAST-GREEDY *runs in $O(n \log^2 n)$ time.*

Proof : The initial stages of the algorithm involve a call to the spanner algorithm in [8], and sorting, which together take $O(n \log n)$ time.

Consider EXAMINE-EDGE. As seen earlier, a call to CHECK-PATH takes constant time. Adding the new inter-cluster edges takes constant time, because there are only a constant number of clusters containing either of the end points of the processed edge. Thus each call to EXAMINE-EDGE takes constant time. EXAMINE-EDGE is itself called $O(n)$ times.

Consider CLUSTER-GRAPH. This procedure first calls CLUSTER-COVER. On partial spanners, the latter can be made to run in $O(n \log n)$ time because there are $O(n)$ edges and each vertex is visited by the SINGLE-SOURCE procedure at most a constant number of times (Lemma 3.5), thus there are overall $O(n)$ heap operations, each taking $O(\log n)$ time. The CLUSTER-COVER procedure can simultaneously build the intra-cluster edges of the cluster-graph. Inter-cluster edges are built in two stages. In the first stage, the SINGLE-SOURCE procedure is run from every cluster center, and in a manner similar to the proof of Lemma 3.5, we can show that each vertex is visited at most a constant number of times. This stage therefore takes $O(n \log n)$ time. In the second stage, each edge (u, v) of G' is checked; there are only a constant number of clusters containing u or v, thus this stage can be processed in linear time. Thus each call to CLUSTER-GRAPH takes $O(n \log n)$ time. CLUSTER-GRAPH is itself called $\log n$ times.

Hence the time complexity of FAST-GREEDY is $O(n \log^2 n)$. □

5 Conclusions

In this paper we show how a simple and natural graph clustering technique can be effectively used in speeding up an existing greedy algorithm for constructing Euclidean graph spanners. We conclude by listing some open problems.

1. Is it possible to reduce the running time to $O(n \log n)$? This may require doing some sort of clustering which *exploits* earlier clusterings. One problem with such an approach is that the error introduced in path length estimations (Lemma 3.3) may multiply beyond a constant factor.

2. The original greedy algorithm has been successful in producing small weight spanners for arbitrary weighted graphs [4]. There exist other spanner algorithms for general graphs which are faster, but none produce spanners with smaller weight. It would be interesting if we can efficiently implement the greedy algorithm for arbitrary weighted graphs. Of course, we would not be able to exploit several nice properties which arise in the geometric case, such as constant degree cluster-graphs.

3. The conjecture in [5] that for any dimensions the weight of the spanner produced by the greedy algorithm is $O(wt(MST))$ needs to be resolved. If proven true, it is likely to imply that asymptotically optimally weighted spanners in any dimensions can be computed in $O(n \log^2 n)$ time.

4. The clustering technique used in this paper is simple and natural; it would interesting to see whether it has other applications in problems involving geometric graphs, especially in speeding up existing algorithms.

References

[1] I. Althöfer, G. Das, D.P. Dobkin, D. Joseph, J. Soares: On Sparse Spanners of Weighted Graphs: Discrete and Computational Geometry, 9, 1993, pp. 81-100.

[2] B. Awerbuch, D. Peleg: Sparse Partitions: IEEE Foundations of Computer Science, 1993.

[3] E. Cohen: Fast Algorithms for Constructing t-Spanners and Paths with Stretch t: IEEE Foundations of Computer Science, 1993.

[4] B. Chandra, G. Das, G. Narasimhan, J. Soares: New Sparseness Results on Graph Spanners: to appear in International Journal of Computational Geometry and Applications.

[5] G. Das, P. Heffernan, G. Narasimhan: Optimally Sparse Spanners in 3-Dimensional Euclidean Space: ACM Symposium on Computational Geometry, 1993, pp. 53-62.

[6] G. Das, P. Heffernan: Constructing Degree-3 Spanners with Other Sparseness Properties: International Symposium on Algorithms and Computations, 1993.

[7] C. Levcopoulos, A. Lingas: There are Planar Graphs Almost as Good as the Complete Graphs and as Short as the Minimum Spanning Trees. Symposium on Optimal Algorithms, LNCS, Springer-Verlag, 1989, pp. 9-13.

[8] J.S. Salowe: Construction of Multidimensional Spanner Graphs with Applications to Minimum Spanning Trees: ACM Symposium on Computational Geometry, 1991, pp. 256-261.

[9] J. Soares: Graph Spanners: Ph.D Thesis, Univ. of Chicago Technical Report CS 92-14, 1992.

[10] P.M. Vaidya: A Sparse Graph Almost as Good as the Complete Graph on Points in K Dimensions: Discrete and Computational Geometry, 6, 1991, pp. 369-381

Dynamic Maintenance of Maximas of 2-d point sets

Sanjiv Kapoor
Department of Computer Science
Indian Institute of Technology
Hauz Khas, New Delhi 110016.
e-mail: skapoor@cse.iitd.ernet.in

Abstract

This paper describes an efficient scheme to dynamically maintain the set of maximas of a 2-d set of points. Using the fact that the maximas can be stored in a Staircase structure, we use a technique which maintains approximations to the Staircase structure. We first show how to maintain the maximas in $O(logn)$ time per insertion and deletion when there are n insertions and deletions. $O(logn)$ is charged per change for reporting changes to the structure. We also show another scheme which requires $O(logn)$ amortized time per insertion and deletion with an output complexity of $O(r)$ steps when r maximal points are to be listed. The data structures require $O(n)$ space.

1 Introduction

Given a set S of points in the $x-y$ plane, $p_i = (x_i, y_i) \in S$ is a maximal point iff it is not dominated by another point p_j where a point p_j dominates p_i iff $x_i < x_j$ and $y_i < y_j$. The set of maximal points of S form a structure which monotonically decreases in the y direction as the x coordinate of the points increases. Such a structure is called a Staircase structure. The maxima problem occurs in a large number of applications in statistics, economics, operations research etc. The reader is referred to the book by Preparata and Shamos for further details [PS]

In this paper is described the design of a data structure for the dynamic maintenance of the maximas of a point set. We maintain approximations to the set in a structure which is an approximate staircase structure. For the static case it has been shown that the staircase structure can be computed in $O(nlogn)$ time eliminating dominated points[KLP]. In the dynamic case however one needs to keep track of the dominated points.

In this case Overmars and Van Leeuwen [OL] have designed a data structure which requires splitting and merging balanced trees when points are inserted and deleted. In their scheme $O(log^2n)$ operations are required for each insertion and deletion. A scheme by Willard and Luekar [WL] gives a bound of $O(logn)$ for updates but the set of maximas is not maintained. Fredrickson and Rodger [FR] and Janardan [J] have designed a scheme which maintains the staircase structure of the set of maximas and allows for insertions and deletions in $O(logn)$ and $O(log^2n)$ operations, respectively. These are the most asymptotically efficient schemes known. Their data structure requires $O(n)$ space. Also [HS] gives a scheme for offline maintenance.

We first present an improved data structure that maintains the staircase structure in $O(logn)$ time per insertion and deletion. A point can be tested to be a maxima in $O(logn)$ time in our representation. And the staircase structure representing the set of Maximas can be listed out in time linear in the size of the set of Maximas but with $O(logn)$ operations to be spent on changes in the report of the maximal elements. The data structure is simple and requires $O(n)$ space.

Furthermore it is shown that the data structure can be modified to give a bound of $O(nlogn + mlogn + r)$ operations when there are n insertions, m deletions and r maximal points are reported.

We hope that the methodology that we have described may also be applicable to dynamic maintenance of other geometric structures, i.e. convex hulls, intersection of half planes and kernel of simple polygons. Since the optimal dynamic maintenance of these geometric structures has been unsolved for some time, the solution in this paper may be of interest in terms of technique.

2 Outline of the Paper.

This paper first describes the structural changes required for maintaining the maximas under insertions and deletions. We then show how to implement the scheme in a data structure requiring $O(n)$ space.

The underlying data structure that we use for storing the points is the red-black tree [T]. It is called the Max-

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

ima Balanced Tree. The set of points, S are stored at the leaves of a red-black tree, which we denote by $MT(S)$, in order of increasing x-coordinate values. At each node is stored the set of maximas of a set of points stored in the subtree rooted at that point. This set is not stored explicitly as we will see. The initial tree can be constructed in $O(nlogn)$ time easily. Let $M(q)$ be the set of maximas at node q in the tree. Furthermore assume that at each node the maximas are specified in a staircase sorted order with points having decreasing y-coordinate and increasing x-coordinate. We let $Top(M(q))$ represent the point with the maximum y-coordinate in the staircase structure representing the maximas at node q. We let $y(p)$ refer to the y-coordinate of point p.

The first scheme that is proposed uses the fact that during insertions binary search is performed only once after which the inserted point is dominated. This fact has also been used in [FR], [J] in their data structure. We thus obtain a solution with $O(logn)$ insertion time and $O(log^2n)$ deletion time. Also isolated is the reason for the $O(log^2n)$ behaviour of the scheme. This scheme is briefly described in section 3.

In section 4, the algorithm is refined so that both insertions and deletions can be done in $O(logn)$ time. We do so by maintaining approximations to the staircase structure. This allows the use of only a constant number of binary searches to update the balanced tree structure storing the maximas. The improvements are described without a description of the secondary structures required to store the maximas at the nodes of the balanced binary tree. Finally, in section 5, the space requirements are shown to $O(n)$. In section 6 the scheme is modified to list the maximal points in linear amortized time.

3 An $O(logn)$ insertion and $O(log^2n)$ deletion scheme.

Let us first consider insertions. Let p be the point to be inserted into the tree $MT(S)$ storing the set of maximas of the point set S. Since the leaves are arranged in x-sorted order a binary search gives us the leaf at which the point p is to be inserted. Let P be the path from the root to the leaf. Suppose, on the path P, we encounter a triple of nodes (u, v, w) where v is the common parent of u and w. u is the left child and is referred to by $left(v)$ and w the right child and referred to by $right(v)$. P uses v and one of u and w.

Firstly suppose the path P uses w. Suppose the maximas of the points in the subtree at w has been recomputed because of the insertion of the new point p. $M(v)$ has to be recomputed from $M(u)$ and $M(w)$ by a merger. The merger, in general, involves locating $Top(M(w))$ in $M(u)$ and then replacing the portion of $M(u)$ dominated by $M(w)$ by $M(w)$ itself. However when a single point, p, is inserted $M(w)$ changes only locally. The merger operation can now be accomplished by changes to the current staircase structure instead of complete recomputation. There are a number of cases depending on the topmost node of $M'(w)$, $Top(M'(w))$ where $M'(w)$ is the updated staircase structure at w. We assume that $Top(M'(w))$ has changed. If it does not, then no change in the merged staircase structure is required.

Case 1: $Top(M'(w))$ is above $Top(M(v))$. In this case the merger requires constant time since $M'(v)$ is actually $M'(w)$. (Fig. M1)

Case 2: $Top(M'(w))$ is not above $Top(M(v))$. In this case the merger is done by a binary search on $M(v)$ and requires $O(logn)$ time. Note that from this node upwords to the root the top node of any staircase structure does not change and new merger points need not be recomputed.(Fig. M2)

Next consider the case when P uses u. Again, suppose $M(u)$ has been modified and let the modified structure be $M'(u)$. There are two cases:

Case 1: $Top(M(u))$ has changed. There are two cases. If $Top(M(w)) < Top(M'(u))$ then change the top node of $M(v)$ to $Top(M'(u))$ if necessary. Otherwise, change $Top(M(v))$ to $Top(M(w))$ if necessary.

Case 2: Top node of $M(u)$ has not changed. In this case we have to determine the changes that are to be made to $M(v)$ since the top of $M(w)$ has to be positioned with respect to the changes in $M(u)$. We note that this can be done in constant time since only a comparison of $Top(M(w))$ with the point inserted into $M(u)$ and points adjacent to it has to be made, provided that node is in the set of maximas. This information is easily maintained as we proceed up the path P.(Fig. M3)

We thus see that an $O(logn)$ insertion is possible since binary search is performed once only in Case 2 when P uses w.

Next consider the affect of deletions. Again consider a path P from leaf to root and a triple of nodes (u, v, w) on the path. The path may use either u or w. First consider the case when the path P uses w. Suppose the maximal set has been recomputed at w and is $M'(w)$. There are two cases depending on $Top(M(w))$: Case 1, where $Top(M(w))$ changes. This case has two subcases. And Case 2 when $Top(M(w))$ does not change. We consider the subcases.

Case 1.1: $Top(M'(w)) < Top(M(u))$. Then $M(v)$ can be computed in $O(logn)$ time by locating $Top(M'(w))$ in $Top(M(u)$ by a binary search.

Case 1.2: $Top(M'(w)) > Top(M(u))$. Update $Top(M(v))$ in $O(1)$ time since $M(v)$ is $M'(w)$.

Case 2: $Top(M(w))$ does not change. $M(v)$ changes exactly as $M(w)$ is changed in the portion of the staircase common to both $M(v)$ and $M(w)$. In this case the list at v need not be searched but only changed at $Top(M(w))$ to incorporate $M'(w)$.

Next consider the case when the path P uses u. $M(v)$ is obtained from $M'(u)$ and $M(w)$ by a merge operation and requires $O(logn)$ operations.

It is the first sub-case of the first case when the path uses w and u that creates the first problem, i.e. it gives a time bound of $O(log^2n)$ since the merge of $M'(u)$ and $M(w)$ requires $O(logn)$ time due to search for $Top(M(w))$ in $M'(u)$ and there may be $\Omega(logn)$ such merges to be performed. Also $Top(M'(w))$ may be changed at $O(logn)$ nodes along path P. Finally we consider the rebalancing operations. They can be performed in $O(logn)$ time since the balancing requires at most $O(1)$ locations of top nodes of staircase structures.

Next we show how to improve the performance to $O(logn)$ per deletions also.

4 An Improved Solution.

4.1 Outline.

In order to improve the performance we attempt to remove the bottleneck introduced by the repeated mergers that are required on path P when $M'(u)$ and $M(w)$ are merged after a deletion. This is done by performing a constant number of mergers as follows:

Let $Mset(P,u)$ be the set of nodes on the path P with the following property: $Top(M(w_i))$, $w_i \in Mset(P,u)$ intersects the section of the staircase in $M(u)$ that is changed due to the deletion. Note that w_i is an ancestor of u and $|Mset(P,u)| = O(logn)$. Only one merger is required and that is the merger with the maximal set $M(w_k)$ where w_k is the node in the binary tree such that $Top(M(w_k))$ exceeds the top, $Top(M(w_i))$, of all other nodes in the set $Mset(P,u)$. All the other mergers required are deferred by actually constructing the chains so that $M(w_i), w_i \in Mset(P,u)$ is merged with $M(u_i)$, where u_i is the sibling of w_i , by joining $Top(M(w_i))$ to either the node in $M(u)$ at which $M((w_k))$ is merged or to $Top(M(u_i))$. This construction is required for a constant number of sets, $Mset(P,u)$ corresponding to nodes u on the path P. This destroys the staircase nature of the maximal set, creating non-horizontal and vertical lines called "kinks" but the deletion now requires $O(logn)$ steps.

The insertion also requires $O(logn)$ steps. It may so happen that during insertion or deletion the node at which the "kinks" occur changes. The kinks thus shift. However we show that the shift only occurs once on the path and there are only $O(logn)$ edges to be transferred at the kinks.

4.2 Detailed Algorithm.

We first formally define the semi-staircase structure at the nodes.

The **Semi-staircase** structure is a sequence of nodes, simultaneously sorted in increasing x order, and in decreasing y-order such that two adjacent vertices are either connected by a horizontal and vertical Manhattan path or by a straight line (called **kink** line). (Fig. M4)

A **Stairstep** is the horizontal and vertical Manhattan path that connects two adjacent vertices.

The **kink** line (u,v) is an approximation to a staircase structure which connects the two vertices u and v.

We need the following property of kinks which will be proved later.

Property Cover: Every kink line connecting a point u, which is not maximal, to another point v is dominated by a horizontal edge or a kink edge, i.e. there exists a point, c with greater y co-ordinate than u and greater x co- ordinates than u and v.

When the point c is connected by a horizontal line to a point s with y-coordinate equal to or just above that of v, s is called the **kink-cover** of the kink line. Note that such a point need not always exist, e.g. it does not exist when v is the top node of a staircase structure. Note also that we may obtain a sequence of kink lines. The Cover property is also satisfied for each edge of this sequence and hence by the entire sequence.

We next show how to dynamically maintain this semi-staircase structure under insertions and deletions.

4.3 Insertions

We first consider Insertions. For inserting point p let P be the path from leaf to root. As we proceed up the path consider the triple of nodes (u,v,w) such that v is the parent of u and w.

Suppose the path P uses w. The cases are similar to the previous section and involve a binary search with $Top(M(w)$ onto a semi-staircase structure. The first change in the algorithm occurs when $y = y(Top(M(w)))$ intersects a kink line joined to v. In this case a new kink line is introduced from $Top(M(w))$ to v. The second change is when at a node w, $Top(M(v))$ is connected by a kink edge to a node in $M(u)$ and the node to which the kink edge is connected is removed by the insertion of a point. $Top(M(v))$ is then located within the new staircase in constant time by computing the intersection with the stairstep newly introduced. Thus as in the previous section one location of $Top(M(w))$ in a semi-staircase structure is required.

Before detailing the location of $Top(M(w))$ we describe in general the procedure for the location of a point p in a semi-staircase. The procedure is recursive. At any node v in the tree we check if the point is located in the part of the semi-staircase at the left son or at the right son and we recur in the appropriate sub-tree. This check is done as follows: Suppose w and u are the right and left sons at v and $M(w)$ is merged with $M(u)$ by a horizontal edge, h. Then the procedure branches to the left if $y(p)$ is greater than the y co-ordinate of point on h otherwise the procedure branches to the right. Alternatively, if at v there is a kink edge merging the right semi-staircase with the left then the procedure recurs at the left. The procedure stops at a leaf having discovered on the staircase a point with y co-ordinate just greater than $y(p)$. This is the node to which p is located. The procedure requires $O(logn)$ steps. Also those $O(logn)$ kink edges on the search path which intersect $y = y(p)$ are obtained.

To update the staircase structures after the location of a point is done, suppose the node to which the location is done is q'. Firstly the edge joining $Top(M(w)$ to q' is added to the semi-staircase structure at v. Next the kink edges intersected by $y = y(s)$ have their top point changed to the node in the staircase just below q' if such a node exists. Else the kink edges become horizontal edges.

When path P uses u, the update after an insertion is again in constant time as described in the previous section.

Thus insertions can be done in $O(logn)$ steps.

4.4 Deletions

Next consider deletions: Again consider the triple (u, v, w). Suppose P, the path from the leaf deleted (corresponding to point q) to the root, uses w. We consider the case depending on whether $M'(w)$ has it's topnode different from $M(w)$ or not.

Case 1.1: $Top(M'(w))$ is changed, i.e. the topmost node of the previous maximal set has been deleted. In this case $Top(M'(w))$ is to be located again in $M(u)$ to construct $M(v)$. There are two subcases: Either the new topmost node is the topmost node of $M(v)$. In this case the check is done in $O(1)$ time. On the other hand the topmost node has to be relocated within $M(u)$ in $O(logn)$ time. However we need to do this only when the set of maximas is affected. Before we consider the number and type of such relocations required we describe in detail one such relocation: There are two possible situations depending on whether the previous topmost node is connected by a kink edge or not.

We first consider the situation when the topmost node in $M(w)$ is not connected by a kink edge to a node in $M(u)$ but by a horizontal edge. Let q' be the node in $M(u)$ to which the topmost node in $M(w)$ was connected. The relocation depends on whether or not other nodes were connected by kink edges to the node q. If no other nodes were connected then the relocation is simple. The relocation is done onto a staircase structure $M(u)$. When kink edges are present at the node q, relocation of the topmost node may destroy the semi-staircase structure and changes have to be made to preserve the semi-staircase structure and the cover property. We describe these changes next.

When other kink edges are present relocating $Top(M'(w))$ involves two sub-cases. Either the top node of $M'(w)$ is relocated below the kink edge (s, q), where s is the point at the first descendant node of u attached to q by a kink edge. Or onto the kink edge. Let d be the node at which s is attached to q by a kink edge. Note that $s \in M(right(d))$ where $right(d)$ is the right son of d. s is located in the staircase structure at $left(d)$. Suppose the node to which the relocation is done is q'. Next, all the kink edges intersected by $y = y(s)$ have their top point changed to the node in the staircase just below q' if such a node exists. Also $Top(M'(w))$ is relocated in the changed structure $M'(u)$. All the operations require $O(logn)$ time since atmost two locations are done and atmost $O(logn)$ kink edges have their top point changed.

In the second sub-case, i.e. when $Top(M'(w))$ is located onto the kink edge, $Top(M'(w))$ is located in $M(u)$ by a procedure similar to the one which locates s in the above case. And all the kink edges intersected by the horizontal edge are changed. In this case too $O(logn)$ steps are required.

Next consider the situation when $Top(M(w))$ was connected by a kink edge to a node q in $M(u)$. Again, let s be the point first attached to q by a kink edge at a descendant node of u (if such a node exists). Either the y-coordinate of $Top(M'(w))$ is less than that of s or not. Firstly consider the case when $Top(M'(w)) < y(s)$. Then $Top(M'(w)$ is relocated in the staircase structure at $M(u)$. Furthermore s may be located also, modifying kink edges intersected by $y = y(s)$ as in the above case. Next, consider the case when $Top(M'(w)) > y(s)$. Then a kink edge from $Top(M'(w))$ to q may be retained in the merge step or $Top(M'(w)$ may be located in the staircase structure modifying the kink edges from s and other nodes as described in the previous paragraph.

We next characterize when the relocation of points in the semi-staircases is necessary: Let $Uset$ be the set of nodes, whose Top has changed due to the deletion of the particular node. Note that the Top value of these nodes is in increasing order of y-coordinate. A relocation is done at the node, say t, with the highest Top value. Furthermore, consider a node $u, u \neq t$ and $u \in Uset$. u is not relocated but $Top(M(right(u)))$ is joined to $Top(M(left(u)))$ by a kink edge.

Case 1.2: $Top(M'(w))$ has not changed. In this case no changes are required in the merger step and the composition of $M'(w)$ and $M(v)$ is carried upwords in $O(1)$ time.

Next suppose the path P uses u. We assume by induction that $M'(u)$ has a semi-staircase structure.

Case 2.1: $M(w)$ intersects $M(u)$ at the stairstep associated with the removed node. Let $R(q)$ be the semi-staircase that replaces the node q deleted. $R(q)$ is obtained as we traverse up the path P. We let S be the set of nodes on the path P such that if $v \in S$ then $Top(M(v)$ intersects the stairstep associated with q and thus $R(q)$. Let w be the node in S such that $Top(M(w)) > Top(M(u)), u \in S$. Then $Top(M(w)$ is located in $R(q)$ to give a node z which forms the node above $Top(M(w))$ in the semi-staircase structure at the parent of w. Otherwise a direct line (kink line) is drawn from $Top(M(u)), u \in S$ to a node t in $R(q)$ where t is the node in $R(q)$ with y-coordinate just less than z and greater than $Top(M(w))$. If such a t does not exist then either a normal staircase is constructed with z being the node next in the semi-staircase structure at the parent of u. Also if t is not present in $M(w)$, where w is the sibling of u, then $Top(M(u))$ is connected via a kink line to $Top(M(w))$. Note that this procedure involves delaying the construction of the straight lines until z is obtained. All the merges required can be done in $O(logn)$ time. (Fig. M5)

Case 2.2: $M(w)$ does not intersect $M(u)$ at the stairstep associated with the changed node. Then no change in the merger need be done.

The above cases complete the description of the changes required after a deletion. It is easily seen that $O(logn)$ steps are required. Furthermore, balancing requires $O(logn)$ steps as each rotation or double rotation requires at most $O(1)$ relocations of topmost nodes giving a bound of $O(logn)$ steps.

Note however that at the end of the deletion some kinks may be left on the final staircase structure at the root.

We next show that the insertion and deletion procedures are correct. It suffices to show that a semi-staircase is maintained at each of the nodes and also that the cover property is maintained.

Lemma 1 *The insertion and deletion procedure maintains semi- staircases at each node of the Maxima Balanced Tree. Moreover the semi-staircase structures satisfy the cover property.*

Proof: The proof is by induction. It is trivially true when there is only one point. Thus assume that there are n points and the data structure contains lists which are staircase structures.

If an insertion is made the operation only introduces a kink edge when the inserted node is located onto a kink edge in the staircase structure. The new kink edge added maintains the semi-staircase structure and also the cover property since either the kink edge is the dominating edge or the dominating edge is unchanged.

When a deletion is performed, the operations firstly introduces kink edges by finding the node closest to the root which intersects the modified staircase and locating it in the staircase structure. All other edges added at descendant nodes are kink edges and valid, i.e. joining a point in the right subtree to a point in the left subtree unless the point is at the top.

These also include kinks introduced by adding edges from the Top of staircases at the left son to the Top of staircases at the right sons. The cover property is maintained since each kink edge (u, v) is covered if it has been determined if u is not in the maximal set. Also when kink and cover edges are modified by locations the semi-staircase structure and cover property are maintained as detailed by the construction procedure. This is also true after balancing of the tree. ■

Since the cover property is maintained at the nodes of the tree kinks in the output are important in that these kinks have to be removed to obtain the actual output. These kinks can be removed at the time of reporting along with other kinks that are discovered when the kinks are removed resulting in an output complexity of $O(m + (chng)logn)$ when there are $chng$ changes in the set of maximas to be reported. Details of the reporting will be discussed in the next section. Once removed these kinks do not occur again in the output.

To summarize

Lemma 2 *The structural changes required to maintain the semi-staircase structures at each node of the Maxima Balanced Tree under insertions and deletions can be carried out in $O(logn)$ operations. Moreover $O(logn)$ operations are required for every change to be reported in the maximal set.*

4.5 An $O(n)$ Space Solution

In this section we show how to implement the scheme for insertion and deletion in $O(n)$ space. This is accomplished by maintaining $O(1)$ information at the nodes of the balanced binary search tree.

Firstly at each node, v, is maintained left and right pointers, $left(v)$ and $right(v)$, to the sons of the node, say u and w. We also maintain two variables TOP and $INTER$ which contain points. TOP is the topmost point in the semi-staircase structure at v (This point is also referred to by $Top(M(w))$ before). And $INTER$ is the node at which $Top(M(w))$ is located in the semi-staircase structure at u, i.e. $INTER$ is the point in the staircase structure at v just above the point TOP at node w in the staircase structure at v. Note that $Top(M(w))$ may be attached to $INTER$ by a kink edge. In the description below the value of $INTER$ for nodes where a kink line is drawn is assumed to be the y-coordinate of the upper point of the kink line. To list the points more efficiently we need another variable *CHANGE* which we detail below.

We next describe the listing and search procedures on the data structure without using *CHANGE*.

4.6 Listing the maximas

To list all the maximas we proceed as follows: We start at the root. At the root node *TOP* gives the first node in the staircase structure. To obtain nodes contributed by the left subtree we test if the topmost node of the right subtree, w, is located strictly below the topmost node of the left subtree, u. If so then we list, recursively, those maximal elements in the subtree at u not yet reported and with y co-ordinates greater than the y-coordinate of $TOP(right(w))$. We then list the maximal elements in the subtree rooted at w recursively. When we apply this procedure recursively we note that at a node the left son may be merged with the right son using a kink edge.

We thus first remove this kink edge before recurring. This procedure is formalized as follows:

```
LIST(v, stop, last);
(This procedure lists the maxima in the
subtree at v with y-cordinate greater
than stop. last is the y-coordinate
of the last maximal point output.)
Begin
    If $last > y(TOP) \geq stop$ then
      begin output TOP; last := TOP end;
    else
    begin
    if (right son is connected to left
     son by a kink edge) then
        LOCATEY( $Top(right(v))$);
     if $y(TOP) > y(INTER)$ then
      begin
        $LIST(left, y(INTER), last)$;
        $LIST(right, stop, last)$
      end
     else $LIST(right, stop, last)$
    end
end.
```

We now describe the implementation of the locate procedure, LOCATEY. The locate algorithm has been described before.

Suppose we are at a node v and wish to search for the horizontal location of p in the semi-staircase structure at v. There are two cases: Suppose $y(INTER)$ is less than $y(TOP)$. If $y(p) \geq y(INTER)$ then we recursively search in the left subtree. And if $y(p) \leq y(TOP(w))$ where w is the right son of the current tree node then we recur to the right. Otherwise we get two subcases. If *INTER* is *TOP* because of a kink edge from the right son to the left son then locate $y(p)$ in the left subtree again. Otherwise we return with *INTER* as the node required.

Note that this procedure requires $O(logn)$ time. A list of kink edges along the path s.t. $y = y(p)$ intersects the kink edge is also obtained. These edges are modified so that their top end is $y(p)$ thus modifying the staircase structure as described before.

The procedure, *LIST*, above is invoked at the root with *stop* being the least y- coordinate. We omit the proof of correctness. The procedure requires time proportional to $O(mlogn)$ where m is the number of maximal points.

To list the maximal points in $O(m)$ time we make the following modification to the data structure: At each node, say v, we store a pointer called *CHANGE*. *CHANGE* points to the first preorder descendant node where a change to the staircase structure at v is made. If $INTER(v) < TOP(v)$ then *CHANGE* points to the first node , say u, in a preorder scan of the left subtree rooted at v where $TOP(right(u)) > INTER(v)$. If $TOP(v) = TOP(right(v))$ then $CHANGE(v)$ is the same as $CHANGE(right(v))$.

It is easy to see that using these pointers we can recursively list the maximas in linear time. This includes the time required to remove kinks along the staircase structure. The following scheme does so:

```
LIST(v, stop, last);
(This procedure lists the maxima in the subtree
at v with y-cordinate greater than stop)
Begin
    If $last > y(TOP) \geq stop$ then
      begin output(TOP); last:= TOP end;
    else
    begin
    if (right son is connected to left
      son by a kink edge) then
        LOCATEY( $Top(right(v))$) and update
         CHANGE using LOCATE CHANGE.
     if $y(TOP) > y(INTER)$ then
      begin
        $LIST(CHANGE, y(INTER), last)$;
        $LIST(right, stop, last)$
      end
     else $LIST(CHANGE, stop, last)$
    end
end.
```

Note that the procedure *LOCATE CHANGE* is invoked to find the value of *CHANGE* at every node. This is required since this value is not updated at every node during the deletion operation when kink edges are constructed. The procedure *LOCATE CHANGE* will be described later in the section. Before *LOCATE*

$CHANGE$ is invoked the kink is removed by locating $TOP(right(v))$. This requires $O(logn)$ steps for every kink removal. Since the corresponding point is not in the previously reported output (else the kink would not be present) the complexity of the procedure is $O(m + chng(logn))$ where m is the number of maximas and $chng$ is bounded by the number of changes in the set of maximas since the last report where every change involves removal of a kink.

We next show that the pointers $CHANGE$ can be assigned at each node initially in $O(1)$ amortized time. This requires the use of a stack to maintain a list of nodes at which relevant changes to the current staircase structure have taken place. The top of the stack gives a candidate to be assigned to $CHANGE$ at the current node, say v, as we traverse up the tree to the root. It is the first descendant node in preorder at which a change in the staircase structure takes place. Note that at this node, say u, $Top(right(u))$ was located within $M(left(u))$. The next item in the stack has the same property, i.e. it is the first descendant node v of the current node with the property that $INTER(v)$ is greater than the value of $INTER$ at the current node. The stack contains nodes, v, such that $INTER(v)$, is in increasing order for the nodes as we go from top to bottom in the stack.

To obtain a stack and to assign a value to $CHANGE$ at the current node, v, we adopt the following procedure. At node v, $TOP(right(v))$ is located within the staircase structure at $left(v)$. If the location is such that $TOP(right(v))$ exceeds $TOP(left(v))$ then the value of $CHANGE$ is the same as the one at $right(v)$ and the stack at the current node is the same as the stack on the right son. If $TOP(right(v))$ is below $TOP(left(v))$ then $CHANGE$ is obtained from the stack of $CHANGE$ values at $left(v)$ as follows: Let Can be the node on top of the stack. If the y-coordinate of $INTER(v)$ is $\geq$ y-coordinate of $INTER$ at Can then Can is removed from the stack and a new candidate is obtained from the top of the stack. This removal is repeated until we find a node on the stack at which the y-coordinate of $INTER$ is above the y-coordinate of $INTER(v)$. $CHANGE$ at v is then assigned to be this node. Moreover the modified stack obtained from the left son is updated with v and the stack at $right(v)$ is removed from further consideration. The initialization of $CHANGE$ can be done in constant amortized time at the nodes since once a node is deleted from a stack it is not considered again.

We finally describe the procedure $LOCATE$ $CHANGE$. This procedure is used when the merge of the right son with the left son at v is done by a non-kink edge using $LOCATEY$. In this procedure let y_p be the y-coordinate of the located point. A candidate, say Can for $CHANGE$ is obtained from the value of $CHANGE$ at the staircase structure at the left son of v. If $y_p \geq y(INTER(Can))$ a new value of Can is obtained from the $CHANGE$ value at the node currently assigned to Can and the process repeated until we find a node at which $INTER$ has greater y-coordinate than y_p. Such a node exists when this procedure is called and $CHANGE$ is assigned this node. At each step the height of Can decreases. Thus this procedure requires $O(logn)$ steps.

4.7 Searching for Dominance

In this section we show how to determine if a given point, p, is dominated by the set of maximas or not. We do so by a search for the y-coordinate of the given point in the set of points with greater x-coordinates. Let P be the path from the root to the leaf where the x-coordinate of the point p is located. Let R be the set of roots of the subtrees to the right of path P. It suffices to check if $y(p) > y(TOP(v))$ for all $v \in R$. If it is, then the point is not dominated otherwise it is.

4.8 Insertions and deletion

To bound the time for insertions and deletions in the data structure we note that the crucial step while performing insertions and deletions is that of locating the topmost node of a semi-staircase structure in another semi-staircase structure. This is problem $LOCATEY$. An $O(logn)$ solution to this problem has been presented above in the $O(n)$ space data structure.

Moreover the $O(1)$ rotations required can also be done in $O(logn)$ steps since they require $O(1)$ mergers, each requiring $O(logn)$ steps. The variables TOP and $INTER$ stored at the nodes at which relocations are performed can also be modified in $O(logn)$ steps. Note that the value of $INTER$ can be obtained by using $LOCATEY$. The value of TOP is immediately available.

We would also like to update the value of $CHANGE$ at the nodes. We consider the insertion path $P(p)$ for a point p. Let c_p be the node at which p is located within a staircase structure. Below that node, p is at the TOP of the staircase structures. Also let c_u be the node at which p is removed from the staircase structures. Above this node p is no longer on the staircase and will not affect the value of $CHANGE$ at these nodes. The procedure which modifies $CHANGE$ values is similar to the initialization procedure. For nodes below c_p the value of $CHANGE$ is not modified. The value of $CHANGE$ at c_p is computed from the left son of c_p by the procedure $LOCATE$ $CHANGE$ which has been described before. For nodes above c_p upto c_u the value of $CHANGE$ is c_p unless a change in the current staircase structure takes place. Again if the change is below the y-coordinate of p the change is stacked. And if the change is above the

y-coordinate of p then c_p is removed from further consideration and the value of $CHANGE$ need not be changed at this node and above since the value of $INTER$ is unchanged.

Furthermore the entire insertion procedure requires $O(logn)$ steps since at most $O(logn)$ values are stacked or popped while recomputing $CHANGE$ on $P(p)$. And $LOCATE\ CHANGE$ is called only once.

A similar procedure to update $CHANGE$ is used when a point is deleted. Firstly, note that only $O(1)$ locations which require a binary search are performed during a deletion. Thus $O(1)$ $LOCATE\ CHANGE$ calls which requires $O(logn)$ operations are required. When kink edges are added only $O(1)$ work is required to update $CHANGE$. Modification of $CHANGE$ along the deletion path is done using stack manipulations as in the insertion case and requres $O(logn)$ time.

We next show how to rebalance. We consider single rotations only. The procedure is similar for double rotations. Let v be the node at which a rebalancing operation is performed. At the nodes affected by the rebalancing $CHANGE$ is obtained by the procedure $LOCATE\ CHANGE$ described above. Also the rebalancing may affect the value of $CHANGE$ above v. Note that rebalancing changes the value of $INTER$ at two nodes. The procedure for updating $CHANGE$ above v is similar to that for updating $CHANGE$ at nodes above c_p in the insertion case when point p is inserted. Note that the initial nodes to be stacked to compute $CHANGES$ are chosen from the nodes affected by the balancing. Thus updates to $CHANGE$ after rebalancings can be done in $O(logn)$ operations.

We state our final result:

Theorem 1 *The maximas of a set of point can be maintained in a data structure which uses $O(n)$ space and which requires $O(logn)$ operations per update when $O(n)$ points are deleted or inserted. The output complexity is $O(m+(chng)logn)$ when m points are reported and when there have been chng changes since the last report.*

5 An improved amortized Bound

In this section we show how the maximas can be reported in linear amortized time.

To improve the reporting bound we note that the $O(logn)$ factor in the reporting time bound essentially arises because a point repeatedly occurs in the changes in the maximal set, i.e. it occurs, is removed from the maximal set by an insertion, and then re-occurs in the set of maximas. Thus its location in the staircase is to be determined at each step. To remedy this we keep this location as follows. Let p be a point and let v be a node in the tree where the point p first occurs within a semi-staircase, i.e. not at the top of the semi-staircase at v. At this node the point has to be located within a staircase structure. This location, $LOC(p)$, is important since it may need to be redetermined during deletions. So we keep this location with the point. However this location is not updated at every insertion or deletion since only the top points in a staircase are manipulated. Thus when this point is to be output the value of $LOC(p)$ may need to be updated. If the location requires a binary search the time for this search is charged to a point that is inserted or deleted and changes this value. We prove this below:

Lemma 3 *Every update of $LOC(p)$ for $p \in M(S)$, the set of maximas of a point set S, can be charged uniquely to an insertion or a deletion.*

Proof: To prove the lemma, suppose $LOC(p)$ for point p is modified. Let u be the node at which $LOC(p)$ is modified and let v be the son of u on $P(p)$ where $P(p)$ is the path from the leaf to the root. The relocation is required when p is added to the set of maximas. The modification must be due to an insertion or a deletion since the last time p was added to the set of maximas. Since the Top of every node is updated during the insertion procedure, p cannot be at the top of the staircases at v when insertions affect v. Thus p must have become the topmost node of a staircase structure due to a deletion and moreover the top node of the staircase structure at v just prior to this event must be above the topmost node of the sibling else p would have been located. (we are referring here to the deletion procedure described in section 4). Thus if p is to be relocated by a search requiring more than $O(1)$ time the corresponding section of the staircase at w, the sibling of v must have been modified. The location is charged to the deletion or insertion that modified the section. Note that other points which have the same LOC value as p are also affected by the same deletion or insertion. Let this set be called $OTHER$. The points in $OTHER$ are dominated by p since p is maximal and the location of p introduces kink edges from these points to the located point. Moreover the points in $OTHER$ can be ordered by domination since they lie along the path to the leaf where $LOC(p)$ is present in the Maxima Tree. The topmost kink edge becomes important and is removed either when p is deleted or when the maximas are to be reported. Either the first point in $OTHER$ is in the set of maximas or some other point dominating it is. The location at this step is charged to p and points in $OTHER$ associated with the point located at this step.

To summarize the details: With each point, p we associate a set $OTHER(p)$. This is the set of points whose LOC will be affected due to the deletion of p. The relocation of a point occurs when the point becomes a maximal element and there is a kink line from the point in the staircase structure. This happens in two ways. Firstly when the point, p becomes the *Top* point at a node during a deletion of a node which was the old *Top* point of the staircase structure containing p at the node and when $LOC(p)$ has changed. The other event happens when the point, p, is to be located onto a staircase structure on which the deleted point was located. In the first case the location is charged to the deleted point which introduced the modified semi-staircase at that node. Note that only one location is done. Other points connected by kink lines and which are encountered during the location are assigned to the located point, p, by adding them to the set $OTHER(p)$ since these points will be affected due to the deletion of p. In the second case the location of p is charged to the deleted point that forced the location, say q, and the elements of $OTHER(q)$ are assigned to $OTHER(p)$. Thus a deleted point is charged twice, once in the first case and once in the second. The second type of charging is part of the deletion process whereas in the first case a kink is removed from the output and a previous deletion is charged.

Finally, note that the LOC values do not change due to rotations and double rotations. The lemma follows. ■

Note, while maintaining $LOC(p)$, $CHANGE$ values at tree nodes also need to be maintained. This involves computing $CHANGE$ by a call to $LOCATE$ $CHANGE$ and updating $CHANGE$ on the path from the leaf to the root. This is done by a procedure similar to that which updates $CHANGE$ during insertions and requires $O(logn)$ steps. We thus have the following result:

Theorem 2 *The set of Maximas of a set of points in the plane can be maintained in $O(nlogn + mlogn)$ operations when there are n insertions and m deletions. The data structure requires linear space. Moreover, r maximal points can be listed out in $O(r)$ operations.*

6 Conclusions and Acknowledgements.

We have shown efficient schemes for dynamically maintaining the set of maximas of a set of points in 2-dimensions. We do so by maintaining approximate information. This technique can also be applied to maintaining maximas in higher dimensions with less effective results. Maintaining the maximas efficiently under both deletions and insertions in higher dimensions is a challenging problem. This technique has also been applied to convex hulls by the author but the results do not give optimal insertion and deletion time as yet. The results will be reported elsewhere. Other applications of this technique would be interesting. I would like thank A. Dutta and S. Sen for helpful discussions. I would also like to thank MPI-Informatik, Germany for providing support during the final part of the work.

References

[FP] F.P. Preparata and M. I. Shamos, Computational Geometry- An Introduction, Springer Verlag, (1985).

[GR] G.N. Fredrickson and S. Rodger, A new approach to the dynamic maintenance of maximal points in a plane, Discrete and Computational Geometry, 5, 365-374, 1990.

[HS] Hershberger and Suri, Offline Maintenance of Planar Configurations, SODA, 1991, pp. 32-41.

[J] R. Janardan, On the dynamic maintenance of maximal points in the plane, 40(2)(1991) 59-64.

[KLP] H.T. Kung, F. Luccio, and F.P. Preparata, On finding the maximas of a set of vectors, J. Assoc. Comput. Mach. 22, 4 (1975), 469-476.

[OL] M.H. Overmars and J.L. Van Leeuwen, Maintenance of configurations in the plane, J. Comp. System Sc. 23 (1981), 166- 204.

[T] R.E. Tarjan, Updating a balanced search tree in $O(1)$ rotations, IPL 16 (1983), 253-257.

[WL] D.E. Willard and G.S. Luekar, Adding range restriction capability to dynamic data structures, J. Assoc. Comput. Mach. 32, 3 (1985), 597-617.

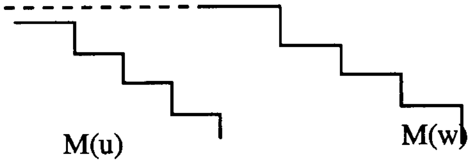


Fig. M1 Merge1

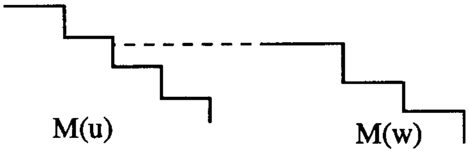


Fig. M2 Merge2

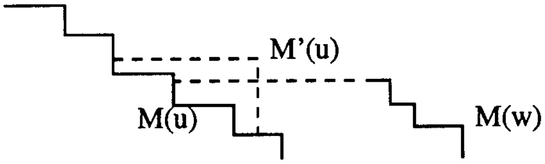


Fig. M3 Delete

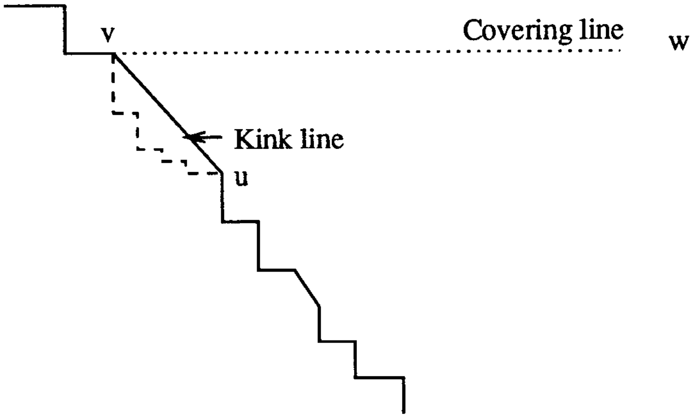


Fig. M4 Semi-Staircase Structure.

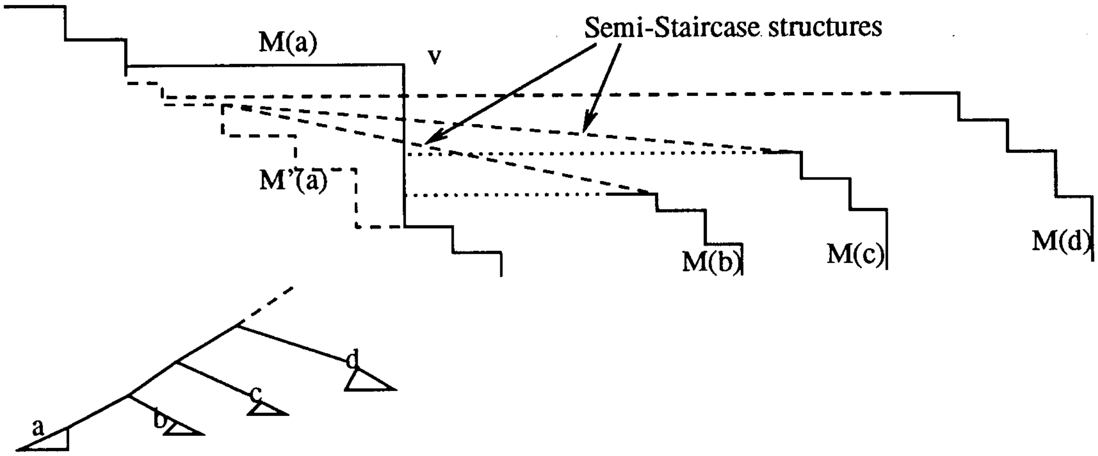


Fig. M5. Deletion of node v.

Biased Finger Trees and Three-Dimensional Layers of Maxima

(Preliminary Version)

MIKHAIL J. ATALLAH*
Department of Computer Sciences
Purdue University
W. Lafayette, IN 47907-1398, USA
E-mail: mja@cs.purdue.edu

MICHAEL T. GOODRICH†
Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218, USA
E-mail: goodrich@cs.jhu.edu

KUMAR RAMAIYER‡
Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218, USA
E-mail: kumar@cs.jhu.edu

Abstract

We present a method for maintaining biased search trees so as to support fast finger updates (i.e., updates in which one is given a pointer to the part of the tree being changed). We illustrate the power of such biased finger trees by showing how they can be used to derive an optimal $O(n \log n)$ algorithm for the 3-dimensional layers-of-maxima problem and also obtain an improved method for dynamic point location.

1 Introduction

Binary search trees are one of the most useful data structures, and are ubiquitous throughout the design and analysis of efficient algorithms [14]. In some cases they serve as a stand-alone structure (e.g., implementing a dictionary or a heap), while in many cases they are used in tandem with other structures, either as primary or secondary structures (or both, as in the range tree [36]). In many dynamic computational geometry algorithms they may even be found as tertiary structures, e.g., Goodrich and Tamassia [19].

1.1 Background and Motivation

When a binary search tree T is maintained dynamically as a primary structure it is appropriate to count, as a part of the update time, the time to perform a top-down search for the node(s) in T being changed. This may not be appropriate when T is used in tandem with other structures, however, for one may be given, as part of the input to an update operation, pointers, or "fingers" [20, 25, 22], directly into the part of T being changed. Such a pointer could come, for example, from a query in some auxiliary data structure. This may, in fact, have been a prime motivating factor behind the method of Huddleston and Mehlhorn [22] for designing a dynamic search tree that has an $\bar{O}(1)$ update time performance for insertions and deletions when the search time is not counted, where we use "$\bar{O}(\cdot)$ time" to refer to a worst-case time bound that is amortized over a sequence of updates.

Another important variant concerns the case when each item i in the search tree is given a *weight*, w_i. This weight is a positive integer that may be proportional to an access probability, as in an optimal binary search tree structure [3, 24]. Or it may represent the size of some auxiliary structure associated with item i, as in a link-cut structure [40] (which itself has many applications [12, 17, 18]) or in a point location structure built using the trapezoid method [8, 35, 38]. In cases with weighted items such as these one desires a search tree satisfying a *bias* property that the depth of each item i in the tree be inversely proportional to w_i. Bent, Sleator and Tarjan [5] give a method for maintaining such a structure subject to update operations, such as insertions, deletions, joins, and splits, as well as predecessor query operations. Most of their update and query operations take $O(\log W/w_i)$ time (in some cases as an amortized bound), with the rest taking slightly more time, where W is the sum of all weights in the tree.

*This research supported by the NSF under Grant CCR-9202807.

†This research supported by the NSF and DARPA under Grant CCR-8908092, by the NSF under Grants CCR-9003299, CDA-9015667, and IRI-9116843.

‡This research supported by the NSF and DARPA under Grant CCR-8908092 and by the NSF under Grant IRI-9116843.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

1.2 Our Results

In this paper we examine a framework for achieving fast finger-tree updates in a biased search tree, and we refer to the resulting structure as a *biased finger tree.* We know of no previous work for a structure such as this. We show that insertions and deletions in a biased finger tree can be implemented in $\bar{O}(\log w_i)$ time, not counting search time, while still maintaining the property that each item i is at depth $O(\log W/w_i)$ in the tree. Moreover, we show that, while split operations will take $\bar{O}(\log W/w_i)$ time (which is unavoidable), we can implement join operations in $\bar{O}(1)$ time.

Our structure is topologically equivalent to that given by Bent, Sleator, and Tarjan [5]. In fact, if each item i has weight $w_i = 1$, then our structure is topologically equivalent to a red-black tree [14, 21, 41]. It is our update methods and amortized analysis that are different, and this is what allows us to achieve running times that are significant improvements over those obtained by Bent, Sleator, and Tarjan, even if one ignores the search times in their update procedures. Moreover, we provide an alternative proof that red-black trees support constant-time amortized finger updates (which is a fact known to folklore).

We show the utility of the biased finger tree structure by giving an optimal $O(n \log n)$-time space-sweeping algorithm for the well-known 3-dimensional layers-of-maxima problem [2, 7, 15, 27]. We also give improved methods for dynamic point location in a convex subdivision [35, 8], and present a method for dynamic point location in staircase subdivision with logarithmic query and update times. We note that although the staircase subdivision we consider is a very restricted form of subdivision, it is the only subdivision we know of for which there is a method achieving logarithmic query and update times. In addition, the optimal algorithm for three-dimensional layers of maxima problem uses our dynamic data structure for staircase subdivision.

2 Biased Finger Trees

Suppose we are given a totally ordered universe U of weighted items, and we wish to represent a collection of disjoint subsets of U in binary search trees subject to the "standard" tree search queries, as well as item insertion and deletion in a tree, and join and split operations on trees (consistent with the total order). Aho, Hopcroft, and Ullman [3] refer to these as the *concatenable queue* operations.

In this section we describe a new data structure that efficiently supports all of the these operations. So as to concentrate on the changes required by an update operation, we will assume that each update operation comes with a pointer to the node(s) in the tree(s) where this update is to begin. Formally, we define our update operations as follows:

Insert(i, w_i, p_{i^-}, T) : Insert item i with weight w_i into T, where p_{i^-} is a pointer to the predecessor, i^-, of i in T (if i has no predecessor in T, then we let this point to i's successor).

Delete(i, p_i, T) : Remove item i from the tree T, given a pointer p_i to the node storing i.

Split(i, T) : Partition T into three trees: T_l, which contains all items in T less than i, the item i itself, and T_r, which contains all items in T greater than i.

Join(T_x, T_y) : Construct a single tree from T_x and T_y, where all the items in T_x are smaller than the items in T_y.

Change-weight(i, w_i', T, p_i) : Change the weight of the item i in T from w_i to $w_{i'}$, given the pointer p_i to the node storing the item i.

Slice(i, T, x, i_1, i_2) : Break the item i in T into two items i_1 and i_2 such that $i^- \le i_1 \le i_2 \le i^+$, and the weight of item i_1 is $x * w_i$ and the weight of item i_2 is $(1-x) * w_i$, where $0 < x < 1$, and i^- and i^+ are predecessor and successor items of i in T respectively.

Fuse(i_1, i_2, i, T) : Combine the items i_1 and i_2 in T into a single item i of weight $w_{i_1} + w_{i_2}$ such that $i_1 \le i \le i_2$ in T. The items i_1 and i_2 need not be siblings, but should be adjacent in the total order.

As mentioned above, our structure is topologically similar to the biased search tree[1] of Bent, Sleator and Tarjan [5]. Our methods for updating and analyzing these structures are significantly different, however, and achieve run times better than those of Bent *et al.* in most cases (see Table 1).

We assume that items are stored in the leaves, and each internal node stores two items, *left* and *right*, which are pointers to the largest item in the left subtree and the smallest item in the right subtree, respectively. In addition, the root maintains pointers to the minimum and maximum leaf items. Every node x of the tree stores a non-negative integer rank $r(x)$ that satisfies the natural extensions of red-black tree rank [41] to weighted sets [5]:

1. If x is a leaf, then $r(x) = \lfloor \log w_i \rfloor$, where i is the item x stores.
2. If node x has parent y, then $r(x) \le r(y)$; if x is a leaf, then $r(x) \le r(y) - 1$. Node x is *major* if $r(x) = r(y) - 1$ and *minor* if $r(x) < r(y) - 1$.
3. If node x has grandparent y, then $r(x) \le r(y) - 1$.

In addition to the above rank conditions, *we also require that a node be minor if and only if its sibling or a child*

[1] Bent, Sleator and Tarjan actually introduce two kinds of biased search trees; our biased finger trees are structurally equivalent to the ones they call *locally biased.*

Update Operation	*Previous Biased Trees [5]*	*Biased Finger Trees*
Search(i,T)	$O(\log W/w_i)$	$O(\log W/w_i)$
Insert(i, w_i, p_{i^-}, T)	$\bar{O}\left(\log \frac{W'}{\min(w_{i^-}+w_{i^+}, w_i)}\right)$	$\bar{O}(\lvert \log \frac{w_i}{w_{i^-}} \rvert + 1)$
Delete(i, T, p_i)	$\bar{O}(\log W/w_i)$	$\bar{O}(\log w_i)$
Split(i, T)	$\bar{O}(\log W/w_i)$	$\bar{O}(\log W/w_i)$
Join(T_x, T_y)	$\bar{O}(\lvert \log W_x/W_y \rvert)$	$\bar{O}(1)$
Change_Weight$(i, w_{i'}, T, p_i)$	$\bar{O}(\log(\frac{\max(W,W')}{\min(w_i,w_{i'})}))$	$\bar{O}(\lvert \log w_i/w_{i'} \rvert)$
Slice(i, T, x, i_1, i_2)	—	$\bar{O}(\log \frac{w_i}{\min(w_{i_1}, w_{i_2})})$
Fuse(i_1, i_2, i, T)	—	$\bar{O}(\min(\log w_{i_1}, \log w_{i_2}))$

Table 1: **Summary of Time Bounds for Biased Tree Operations.** W and W' are the sum of the weights of all the items before and after the update operation respectively, W_x (W_y) denotes the sum of weights in T_x (T_y), and i^- (i^+) denotes the predecessor (successor) of i. All the complexities for update operations only count the time to perform the update, and do not include search times.

of its sibling is a major leaf [5]. We refer to this as the *bias* property.

In the remainder of this section we provide algorithms for various update operations on biased finger trees, and also analyze their amortized complexities.

2.1 Rebalancing a Biased Finger Tree

We begin our discussion by analyzing the time needed to rebalance a biased tree after an update has occurred. We use the *banker's view* of amortization [42] to analyze the rebalancing and update operations. With each node x of a biased finger tree we associate[2] a value, $C(x)$, of "credits", with $0 \leq C(x)$. Moreover, we partition these credits into three types—one type that is similar to those used in the analysis of Bent, Sleator, and Tarjan [5], one type that assigns 1 credit to the nodes on the spine of the tree, and one type suggested by Kosaraju [26]. We omit details here.

After an update operation, we perform promote or demote operations on the ranks of some of the nodes of the biased finger tree, which increase or decrease the rank of the nodes, respectively. These operations locally preserve the rank properties, but may cause violation of the rank property on other nodes, which may require further promotions or demotions or may even require rebalancing to globally preserve the rank properties. We show that the total complexity of promotion, demotion and rebalancing operations due to a single promote or demote operation is $\bar{O}(1)$. The structure of our case analysis follows closely that Tarjan [41] used for red-black trees. We again omit the details here, and in the full version prove the following:

Lemma 2.1 *The total complexity of promotion, demotion and rebalancing operations on a biased finger tree due to a single promote/demote operation is $\bar{O}(1)$ (actually at most 8 credits). Also, each operation adds at most two pairs of equal rank siblings to the tree.*

2.2 Update Operations

We now discuss the methods for various update operations. We begin with the join operation. We describe a "bottom-up" strategy, which, contrasts with the "top-down" approach of Bent, Sleator and Tarjan [5].

Join: Consider the join of two biased trees T_x and T_y. Let u and v be the rightmost leaf and the leftmost leaf of T_x and T_y respectively. Let w and l be the parent of u and v respectively (see Fig. 1). The nodes u and v can be accessed using the pointers in the root nodes x and y respectively. We have the following cases:

Case 1. $r(x) = r(y)$. In this case we create a new node z with T_x as the left subtree and T_y as the right subtree, and we assign a rank of $r(x) + 1$ to z. We then proceed up the tree as in the promote operation.

Case 2. $r(x) < r(y)$. In this case we traverse the rightmost path of T_x and the leftmost path of T_y, both bottom up, in the increasing order of ranks of the spine nodes. As we proceed we store the pointers to the nodes encountered and also tags indicating whether they are from T_x or T_y in an array A ordered by node ranks. We also keep track of the nodes of equal ranks last encountered in the two paths and we terminate the traversal when reaching the root x. Suppose t is the smallest rank node along the leftmost path of T_y having rank greater than x. Suppose a and b are the last encountered equal rank nodes during the traversal, and note that the node x was stored as the last node in array A. We attach x along with its left subtree to

[2] This credit notion is only used for analysis purposes. No actual credits are stored anywhere.

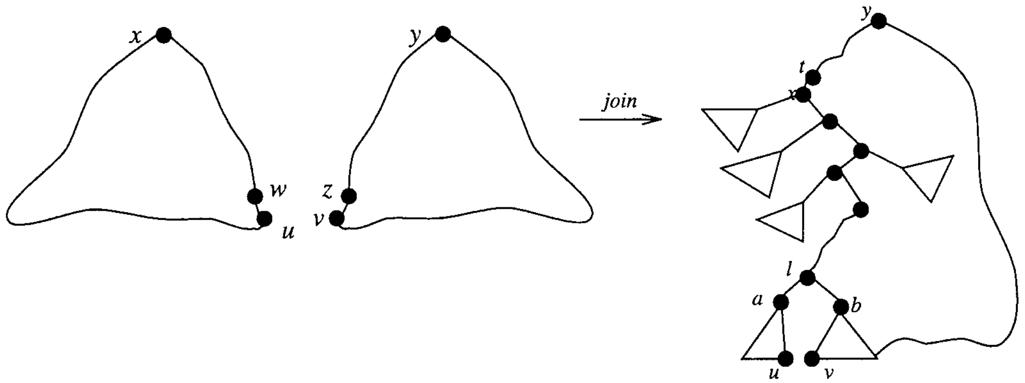


Figure 1: Join of trees T_x and T_y.

t as the left child of t. For the other nodes in A, we proceed as follows (see Fig. 1): Consider the next node, c, in the array A. Suppose c is part of T_x. If the successor of c in A is a node d from T_x, we attach c and its left subtree as a right child of d. If the successor of c in A is a node d from T_y, we attach c and its left subtree as a left child of d. Suppose c is part of T_y. If the successor of c in A is a node d from T_x, we attach c and its right subtree as a right child of d. If the successor of c in A is a node d from T_y, we attach c and its right subtree as a left child of d. We continue this process until the nodes a and b are encountered. Then, we create a new node z with T_a and T_b as left and right subtrees respectively. We assign a rank of $r(a) + 1$ to z and rebalance if required through a promote operation. This terminates the join. If there are no equal rank spine nodes a and b, then we join all the nodes in the array A in the above manner.

Case 3. $r(y) < r(x)$. Symmetrical to above case.

Analysis: We show in the full version that the total number of credits needed to perform this update is $\bar{O}(1)$. This, and Lemma 2.1, establishes that the running time for a join is $\bar{O}(1)$.

Split: We perform the split operation as in [5]. We show that with the same complexity we can preserve all three types of credits in the nodes of the resulting trees.

Insertion: Consider the insertion of an item i with weight w_i to a biased finger tree T. Recall that i^- denotes the immediate predecessor of item i in T if it exists, and immediate successor of i in T otherwise. We provide a pointer to i^-. In the full version we describe how to perform a bottom-up insertion from i^-. The most interesting case occurs when the new item has weight much larger than i^-, for we must then splay i up the tree to its proper position while joining together the trees whose roots become siblings in this process (so as to maintain our minor-node bias property). We show that this can be done in $\bar{O}(|\log w_i/w_{i^-}|)$ time.

Deletion: Consider the deletion of an item i with weight w_i from the biased finger tree T for which a pointer to the item i is provided. This operation is similar to that of insertion and we have different cases based on whether the node deleted is minor/major. We use different operations to preserve bias property in each case. We give details in the full version where we show the complexity of deletion to be $\bar{O}(\log w_i)$.

We summarize:

Theorem 2.2 *One can maintain a collection of biased search trees subject to tree searches, element insertion and deletion, change of weight of element, slicing and fusing of elements, as well as tree joining and splitting, in the bounds quoted in Table 1 for biased finger trees, not counting the search times.*

Bent *et al.* [5] show that repeated single-node joins on the right hand side can construct a biased finger tree in $O(n)$ worst-case time. In our case, however, we can show the following:

Theorem 2.3 *Any sequence of joins that constructs a biased finger tree of n items can be implemented in $O(n)$ worst-case time.*

Proof: The proof follows immediately from the fact that our join algorithm on biased finger trees takes $\bar{O}(1)$ time. ■

Let us now turn our attention to some non-trivial applications.

3 The Layers-of-Maxima Problem

In this section, we use the biased finger tree data structure to solve an open (static) computational geometry

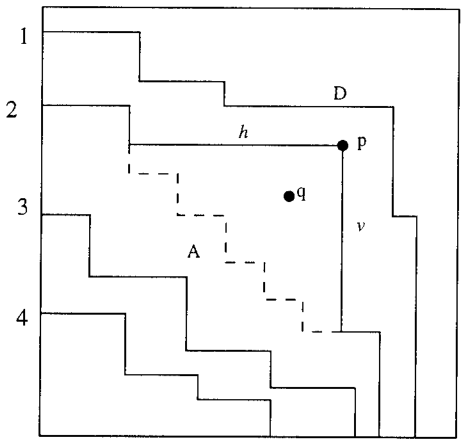


Figure 2: Computation of Layer for a New Point p.

problem: the 3-dimensional layers-of-maxima problem. Before we describe our method, however, we introduce some definitions. A point $p \in \Re^3$ *dominates* a point $q \in \Re^3$, if $x(q) < x(p)$, $y(q) < y(p)$, and $z(q) < z(p)$. Given a set S of points in $\Re^3$, a point p is a *maximum* point in S, if it is not dominated by any other point in S. We define the *dominance region* of a point p in $\Re^3$ as a set $D \subset \Re^3$ such that p dominates every point $q \in D$. The *maxima set* problem is to find all the maximum points in S. Kung, Luccio, and Preparata [27] showed that this problem can be solved in $O(n \log n)$ time. In the related *layers-of-maxima* problem, one imagines an iterative process, where one finds a maxima set M in S, removes all the points of M from S, and repeats this process until S is empty. The iteration number in which a point p is removed from S is called p's *layer*, and we denote it by $l(p)$, and the layers-of-maxima problem is to determine the layer of each point p in S. This is related to the well-known *convex layers* problem [7], and it appears that it can be solved for a 3-dimensional point set S in $O(n \log n \log\log n)$ time [1] using the dynamic fractional cascading technique of Mehlhorn and Näher [32]. We show how to solve the 3-dimensional layers-of-maxima problem in $O(n \log n)$ time, which is optimal [3].

We solve this problem using a three-dimensional sweep, and a dynamic method for point location in a staircase subdivision. Given a set S of n points in $\Re^3$, we first sort the points along the z axis, and then sweep the points in the decreasing order of their z coordinates to construct the maxima layers. When we sweep across a point, we compute its layer using the information about the layers computed so far. The information we maintain for each layer is the union of the dominance regions of the points in that layer. We denote this by $\mathcal{D}(l)$, for a layer l. We show that to correctly compute the layers, it is sufficient to maintain for each layer l, the boundary of the intersection of the sweep plane, say π, with $\mathcal{D}(l)$. The intersection region is two dimensional, and we call its boundary a *staircase.*

The shape of the staircase representing a layer changes, as we continue the sweep. During the sweep, we maintain only a subset of points that belong to each layer. This is because, if points, say p and q, belong to a layer l, and if the projection of dominance region of p onto π dominates that of q, then point q will not be part of the boundary of intersection. This simplifies the identification of layer for a new point. Let $S' \subset S$ be the current set of points maintained by the algorithm. We show that S' has the property that the the staircases corresponding to the layers of points in S', subdivide the xy-plane into disjoint regions (as defined earlier). We call this a *staircase subdivision.* We show that this property is preserved at each step of the sweep, when we compute the layer for a new point. Hence, at any instant the current set of maxima layers form a staircase subdivision, and we reduce the computation of layer for a new point to operations on the staircase subdivision. We call the region in the subdivision between two staircases, a *face.* Hence, if there are m layers, they subdivide the xy-plane into $m+1$ faces. The projection of each new point onto the xy-plane belongs to a unique face among these $m+1$ faces. We work with the staircase subdivision, and the projection of a new point, to identify the point's layer.

The algorithm for computing the layer number of a new point p is, then, as follows:

1. Identify the two staircases in the staircase subdivision between which the new point p lies. Assign p to higher-numbered layer of these two. For example, in Figure 2, p lies between the layers 1 and 2, and gets assigned to layer 2. If p lies below the highest-numbered layer, say m, then assign p to a new layer $m+1$.

2. Compute the horizontal segment h, and the vertical segment v from p, which hit the boundary or some layer (of course the layer hit by h is the same layer that is hit by v, and has same number as p's just computed layer).

3. Insert the segment h and the segment v into the subdivision.

4. Delete the segments in the layer $l(p)$, which are dominated by p in the xy-plane. For example, in Figure 2, we delete segments in portion A of layer 2.

Correctness and Analysis: We now show that each new point p is identified with its correct layer. We use Figure 2 to illustrate the idea. Consider layer 2. The

[3] A simple linear time reduction can be shown from sorting problem to three-dimensional layers-of-maxima problem, thereby showing a lower bound of $\Omega(n \log n)$ for three-dimensional layers of maxima problem.

staircase of layer 2 is the boundary of the intersection of the union of the dominance regions of the points in the layer 2 with the sweep plane π. Since the points are processed in decreasing order of their z coordinates, when we insert p, the dominance region of point p does not dominate any of the points in layer 2. Also the points in layer 2 do not dominate p along x and y coordinates. So, point p belongs to layer 2 (i.e., $l(p) = 2$), as identified by the algorithm. Now to update the boundary of intersection with π, we delete the portion A from layer 2, and include the segments h and v into the subdivision. To see why only the boundary is maintained, we observe that if point q is introduced later, the algorithm will identify q with layer $l(p)$. But q does not belong to layer $l(p)$, since q is dominated by p. So, it should either initiate a new layer, or belong to an existing layer (see Figure 2). In any case, $l(q) = l(p) + 1$. Also, we observe that after deleting points in portion A, the new boundary for layer 2 is in the form of a staircase, thus preserving the property.

Suppose location, insertion and deletion of a vertex/edge in a staircase subdivision take time $Q(n), I(n)$, and $D(n)$ time respectively. We implement step 1 as a point location in staircase subdivision, and it takes $O(Q(n))$ time. We represent the staircase corresponding to each layer by a dictionary, for ordering along x and y axes (since both are same ordering). Using these data structures, we compute in $O(\log n)$ time the horizontal segment h, and the vertical segment v for a new point p. Hence, step 2 takes $O(\log n)$ time. Therefore, the total time for computing the layer of each point is $O(\log n + Q(n) + I(n) + k * D(n))$, where k is the number of points deleted in step 4. Since each point is deleted at most once, and is not inserted back, we amortize the cost of k deletions on each of the k points. Hence, the complexity of computing layer of each new point is $\bar{O}(\log n + Q(n) + I(n) + D(n))$. In the next section, we show a method which achieves $Q(n) = I(n) = D(n) = \bar{O}(\log n)$, resulting in a $\bar{O}(\log n)$ algorithm for computing the layer of a single point. This gives us an $O(n \log n)$ algorithm for computing the three dimensional layers-of-maxima, and is optimal.

4 Dynamic Point Location

In this section, we address the general problem of dynamic point location in a convex subdivision. So, suppose we are given a connected subdivision $\mathcal{S}$ of the plane such that $\mathcal{S}$ partitions the plane into two-dimensional cells bounded by straight line segments. The *point location* problem is to construct a data structure that allows one to determine for any query point p the name of the cell in $\mathcal{S}$ that contains p (see [13, 15, 16, 23, 28, 29, 35, 36, 39]). It is well-known that one can construct a linear-space data structure for answering such queries in $O(\log n)$ time [13, 16, 23, 39].

These optimal data structures are *static*, however, in that they do not allow for any changes to $\mathcal{S}$ to occur after the data structure is constructed. There has, therefore, been an increasing interest more recently into methods for performing point location in a dynamic setting, where one is allowed to make changes to $\mathcal{S}$, such as adding or deleting edges and vertices. It is easy to see that, by a simple reduction from the sorting problem, a sequence of n queries and updates to $\mathcal{S}$ requires $\Omega(n \log n)$ time in the comparison model, yet there is no existing fully dynamic framework that achieves $O(\log n)$ time for both queries and updates (even in an amortized sense). The currently best methods are summarized in Table 2. The results in that table are distinguished by the assumptions they make on the structure of $\mathcal{S}$. For example, a *convex* subdivision is one in which each face is convex (except for the external face), a *staircase* subdivision is one in which each face is a region bounded between two infinite staircases, a *rectilinear* subdivision is one in which each edge is parallel to the x- or y-axis, a *monotone* subdivision is one in which each face is monotone with respect to (say) the x-axis, a *connected* subdivision is one which forms a connected graph, and a *general* subdivision is one that may contain "holes." The interested reader is referred to the excellent survey by Chiang and Tamassia [9] for a discussion of these and other results in dynamic computational geometry.

4.1 Our Data Structure

Suppose we are given a convex subdivision $\mathcal{S}$ that we would like to maintain dynamically subject to point location queries and edge and vertex insertions and deletions. As mentioned above, our method for maintaining $\mathcal{S}$ is based upon a dynamic implementation of the "trapezoid method" of Preparata [35] for static point location. Incidentally, this is also the approach used by Chiang and Tamassia [8], albeit in a different way. Let us assume, for the time being, that the x-coordinates of the segment endpoints are integers in the range $[1, n]$ (we will show later how to get around this restriction using the the $BB[\alpha]$ tree). We define our structure recursively, following the general approach of Preparata [35]. Our structure is a rooted tree, T, each of whose nodes is associated with a trapezoid τ whose parallel boundary edges are vertical. We imagine the trapezoid τ as being a "window" on $\mathcal{S}$, with the remaining task being that of locating a query point in $\mathcal{S}$ restricted to this trapezoidal window. With the root of T we associate a bounding rectangle for $\mathcal{S}$.

[4] Cheng and Janardan's update method is actually a de-amortization of an amortized scheme via the "rebuild-while-you-work" technique of Overmars [34].

[5] Our method can actually used for any dynamic point location environment satisfying a certain *pseudo-edge* property.

Type	Queries	Insert	Delete	Reference
general	$O(\log n \log\log n)$	$\bar{O}(\log n \log\log n)$	$\bar{O}(\log^2 n)$	Baumgarten *et al.* [4]
connected	$O(\log^2 n)$	$O(\log n)$	$O(\log n)$	Cheng-Janardan [11][4]
connected	$O(\log n)$	$\bar{O}(\log^3 n)$	$\bar{O}(\log^3 n)$	Chiang *et al.* [10]
monotone	$O(\log n)$	$\bar{O}(\log^2 n)$	$\bar{O}(\log^2 n)$	Chiang-Tamassia [8]
monotone	$O(\log^2 n)$	$O(\log n)$	$O(\log n)$	Goodrich-Tamassia [18]
rectilinear	$O(\log n \log\log n)$	$\bar{O}(\log n \log\log n)$	$\bar{O}(\log n \log\log n)$	Mehlhorn-Näher [32]
convex	$O(\log n + \log N)$	$O(\log n \log N)$	$O(\log n \log N)$	Preparata-Tamassia [38]
convex[5]	$O(\log n)$	$\bar{O}(\log n)$	$\bar{O}(\log^2 n)$	this paper
staircase	$O(\log n)$	$\bar{O}(\log n)$	$\bar{O}(\log n)$	this paper

Table 2: Previous and New results in dynamic point location. N denotes the number of possible y-coordinates for edge endpoints in the subdivision.

Let v therefore be a node in T with trapezoid τ associated with it. If no vertex or edge of $\mathcal{S}$ intersects the interior of τ, then we say that τ is *empty*, in which case v is a leaf of T. Note that in this case any point determined to be inside τ is immediately located in the cell of $\mathcal{S}$ containing τ. Let us therefore inductively assume that τ contains at least one endpoint of a segment in $\mathcal{S}$. There are two cases:

1. There is no face of $\mathcal{S}$ that intersects τ's left and right boundaries while not intersecting τ's top or bottom boundary. In this case we divide τ in two by a vertical line down the "middle" (we choose a vertical line which balances the height of the tree on both sides) of τ, an action we refer to as a *vertical* cut. This creates two new trapezoids τ_l and τ_r, which are ordered by the "right of" relation. We create two new nodes v_l and v_r, which are respectively the left and right child of v, with v_l associated with τ_l and v_r associated with τ_r.
2. There is at least one face of $\mathcal{S}$ that intersects both the left and right boundaries of τ and does not have a spanning edge of τ as its top or bottom boundary. In this case we "cut" τ through each of the faces of $\mathcal{S}$ that intersect τ's left and right boundaries. This creates a collection of trapezoids $\tau_1, \tau_2, \ldots, \tau_k$ ordered by the "above" relation. We refer to this action as a collection of *horizontal* pseudo-cuts (even though it would be more accurate to call them "non-vertical pseudo-cuts"). We associate a node v_i in T with each τ_i and make this set of nodes be the children of v in T, ordered from left-to-right by the "above" relation on their respective associated trapezoids.

Repeating the above trapezoidal cutting operations recursively at each child of v creates our tree T (see Figure 3). The tree T, of course, cannot yet be used to perform an efficient point location query, since a node v in T may have many children if its associated action forms a collection of horizontal cuts. To help deal with this issue we define the *weight* of a node $v \in T$ to be the number of leaf descendents of v in T, and we use $w(v)$ to denote this quantity. Given this weight function, then, we store the children of each node v in T as leaves in a biased finger tree T_v and doubly link all the leaves of T_v. Of course, such a biased finger tree is a trivial tree for each node v corresponding to a vertical cut, but this is not a problem, for it gives us a way to efficiently search the children of a node whose corresponding action is a collection of horizontal cuts.

The structure of T satisfies an invariant that if a face f spans a trapezoid, then either it has a spanning edge e of the subdivision on its top or bottom boundary or it is split into two by a pseudo-cut. In either case, the face f has a bounding spanning edge if it spans τ. We say that a face f or an edge e of the subdivision *covers* a trapezoid τ if it spans τ horizontally and it does not span any ancestor of τ in T. The structure of T has the property that any face or edge covers at most $O(\log n)$ trapezoids and also each face or edge covers at most two nodes at any level of T. These properties follow easily from segment tree like arguments [31].

We now describe the point-location query algorithm for our data structure. Consider the operation $query(\tau, x, y)$, where x and y represent the coordinates of the query point and τ is a current trapezoid in the subdivision (which represents a node in our primary data structure). We alternately make comparisons with nodes in the primary and secondary data structures. In the primary data structure (triangular nodes representing trapezoids), we compare the x value of the point against the x value of the vertical cut at τ. This identifies the left or right secondary data structure of τ containing the query point. We then use the secondary data structure to identify a trapezoid containing the query point among the several trapezoids separated by horizontal cuts. In the secondary data structure i.e., in the biased finger tree stored in the trapezoid τ, we compare the (x, y) value of the point against the sup-

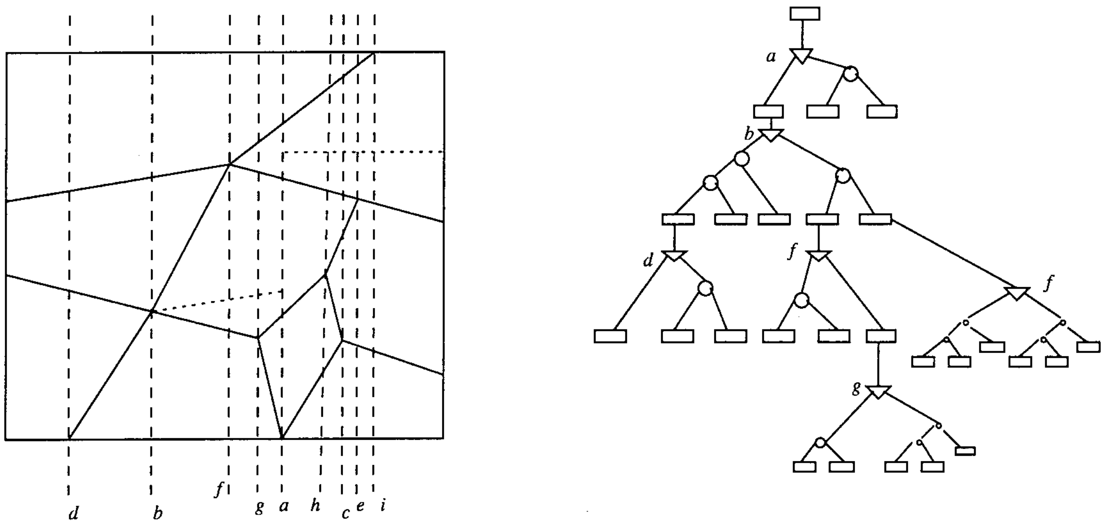


Figure 3: Trapezoidal Decomposition of a Convex Subdivision with Pseudo-Edge Cuts. The triangular nodes denote vertical cuts and the circular nodes denote horizontal cuts.

porting line of the spanning edge or pseudo-cut [6]. This identifies a leaf node, say q, in the biased finger tree that represents a trapezoid, say τ_q, in the primary data structure. We now recursively locate the point by calling $query(\tau_q, x, y)$.

The arguments in the previous section on the structure of our data structure imply that, starting from the root of T, we can perform the point location query in $O(\log w(r) + depth(\tau))$ time in the worst case, where r denotes the root of T. This is because the times to perform the biased merge tree queries down a path in T form a telescoping sum that is $O(\log w(r))$. Noting that $w(r)$ is $O(n \log n)$ [35] and $depth(\tau)$ is $O(\log n)$ (since our primary data structure is kept balanced) gives us the desired result that a point location query takes $O(\log n)$ time.

Our method for updating this structure is rather involved; hence, for space reasons, we can only sketch the main ideas in this extended abstract. In the case of an insertion of an edge e into a face f our method traverses down the tree for the endpoints of e. At each node that f covers and such that e cuts so that the new faces no longer cover we must perform $O(1)$ join operations. Since there are $O(\log n)$ such nodes, the total time for these updates, then, is $\bar{O}(\log n)$. In addition, there are also $O(\log n)$ places in the tree where we must insert this new edge, each of which can be implemented in $\bar{O}(1)$ time, given a pointer to the location for this insertion (which we obtain from the Δ list for f). Thus, the total time for an insertion is $\bar{O}(\log n)$. The case of a deletion is essentially the reverse of the above operations. This operation runs in $\bar{O}(\log^2 n)$ time, however, since we now may have to perform $O(\log n)$ split operations (and their complexity, unfortunately, does not form a telescoping sum).

4.2 Rebalancing the Primary Structure

In this section we show how to relax the constraint that the endpoints have x coordinates in the range $[1, n]$. We use a $BB[\alpha]$-tree as a primary tree for vertical cuts with biased finger tree as a secondary structure in each node. We briefly review the properties of $BB[\alpha]$-tree. Let $f(l)$ denote the time to update the secondary structures after a rotation at a node whose subtree has l leaves. Also, assume that we perform a sequence of n update operations, each an insertion or a deletion, into an initially empty $BB[\alpha]$-tree. Now, we have the following times for rebalancing [30]:

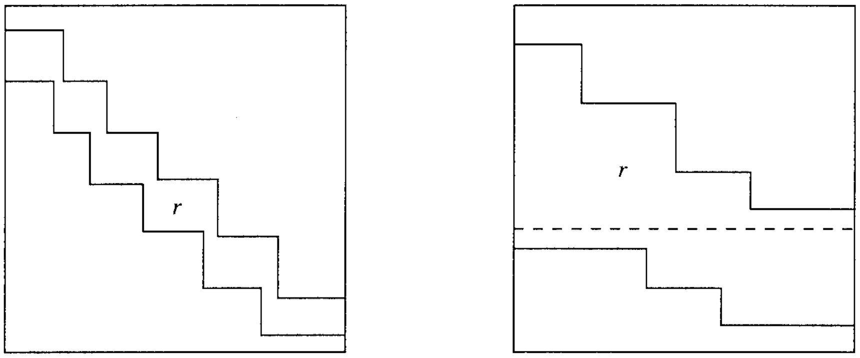
- If $f(l) = O(l \log^c l)$, with $c \geq 0$, then the rebalancing time for an update operation is $\bar{O}(\log^{c+1} n)$.
- If $f(l) = O(l^a)$, with $a < 1$, then the rebalancing time for an update operation is $\bar{O}(1)$.

In our case, we show $f(n) = O(n)$ and so the rebalancing cost is $\bar{O}(\log n)$. We give the details in the full version.

Thus we show,

Theorem 4.1 *Given a convex subdivision S of n vertices, there exists a data structure which allows point location queries in $O(\log n)$ time, vertex/edge insertion in $\bar{O}(\log n)$ time, and vertex/edge deletion in $\bar{O}(\log^2 n)$ time.*

[6] When we use the query algorithm to locate edges, if the edge spans the trapezoid τ then we compare the y-value of the point of intersection of the edge with the left boundary of τ against the y-values of the points of intersections of the horizontal cuts with the left boundary of τ.



a. A Spanning Face Without any Pseudo-cut

b. A Spanning Face With a Pseudo-cut

Figure 4: Pseudo-Cut for a staircase spanning face.

4.3 Dynamic Point Location in Staircase Subdivisions

In this section, we construct an $O(\log n)$ depth trapezoidal structure for staircase subsubdivisions similar to the one for convex subdivision, which allows us to perform query in $O(\log n)$ time, and updates in $\bar{O}(\log n)$ time. We always require that deletion of an edges (vertices) should always be accompanied by an insertion of an edge (vertices).

The trapezoidal structure for staircase subdivision satisfies a slightly different invariant from that of convex subdivision. Here we use an invariant that if a face f spans a trapezoid τ and f can be split into two faces by a (truly) horizontal spanning segment, say e, of τ, then we split τ into two trapezoids using e. So, unlike convex subdivision, not every spanning face of r will have a pseudo-cut (see Figures 4.a and 4.b) here. Using this invariant, we can easily do an analysis similar to that of convex subdivision to show that the insert operation in a staircase subdivision takes takes $\bar{O}(\log n)$ time. The details are omitted.

Using this invariant, we can easily do an analysis similar to that of convex subdivision to show that the insert operation in a staircase subdivision takes takes $\bar{O}(\log n)$ time. The details are omitted. We observe that, each edge deleted is always bounded on the top and on the right by "long" edges. This implies that we already have a horizontal spanning pseudo edge for the face resulting after deletion, and hence deletion of an edge does not require an introduction of a new pseudo-cut into the subdivision. This eliminates the costlier case of deletion which takes $\bar{O}(\log^2 n)$, and all the other cases take $\bar{O}(\log n)$ time. We give details in full version. Thus, this observation results in a $\bar{O}(\log n)$ time method for deletion of an edge in the staircase subdivision.

Thus we show,

Theorem 4.2 *Given a staircase subdivision S of n vertices, there exists a data structure which allows point location queries in $O(\log n)$ time, edge (vertex) insertion in $\bar{O}(\log n)$ time, and edge (vertex) deletion in $\bar{O}(\log n)$ time (the deletions are always coupled with insertions, however).*

The edge insertion and deletions performed on staircase subdivision in our three-dimensional layers of maxima algorithm satisfy the constraints specified in Theorem 4.2. Therefore we have,

Theorem 4.3 *Given a set S of n points in $\Re^3$, one can construct the layers of maxima for S in $O(n \log n)$ time, which is optimal.*

Acknowledgement

We would like to thank Rao Kosaraju for several helpful conversations concerning the topics of this paper.

References

[1] P. Agarwal, private communication, 1992.

[2] A. Aggarwal and J. Park, "Notes on searching in multidimensional monotone arrays," in *Proc. 29th IEEE Symposium on Foundations of Computer Science*, 1988, 497–512.

[3] A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *Data Structures and Algorithms*, Addison-Wesley (Reading, Mass.: 1983).

[4] H. Baumgarten, H. Jung, and K. Mehlhorn, "Dynamic Point Location in General Subdivisions," *Proc. 3rd ACM-SIAM Symp. on Discrete Algorithms*, 1992, 250–258.

[5] S.W. Bent, D.D. Sleator, and R.E. Tarjan, "Biased search trees," *SIAM Journal of Computing*, 14(3):545–568, August 1985.

[6] N. Blum and K. Mehlhorn, "On the Average Number of Rebalancing Operations in Weight-Balanced Trees," *Theoretical Computer Science*, **11**, 1980, 303–320.

[7] B. Chazelle, "On the convex layers of a planar set," *IEEE Trans. Inform. Theory*, **IT-31** 1985, 509–517.

[8] Y.-J. Chiang and R. Tamassia, "Dynamization of the Trapezoid Method for Planar Point Location," *Proc. ACM Symp. on Computational Geometry*, 1991, 61–70.

[9] Y.-J. Chiang and R. Tamassia, "Dynamic Algorithms in Computational Geometry," *Proc. of the IEEE*, **80**(9), 1992.

[10] Y.-J. Chiang, F.P. Preparata, and R. Tamassia "A Unified Approach to Dynamic Point Location, Ray Shooting, and Shortest Paths in Planar Maps," *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms*, 1993, 44–53.

[11] S.W. Cheng and R. Janardan, "New Results on Dynamic Planar Point Location," *31st IEEE Symp. on Foundations of Computer Science*, 96–105, 1990.

[12] R.F. Cohen and R. Tamassia, "Dynamic Expression Trees and their Applications," *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, 1991, 52–61.

[13] R. Cole, "Searching and Storing Similar Lists," *J. of Algorithms*, Vol. 7, 202–220 (1986).

[14] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, MIT Press (Cambridge, Mass.: 1990).

[15] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, NY, 1987.

[16] H. Edelsbrunner, L.J. Guibas, and J. Stolfi, "Optimal Point Location in a Monotone Subdivision," *SIAM J. Computing*, Vol. 15, No. 2, 317–340, 1986.

[17] D. Eppstein, G.F. Italiano, R. Tamassia, R.E. Tarjan, J. Westbrook, and M. Yung, "Maintenance of a Minimum Spanning Forest in a Dynamic Planar Graph," *J. Algorithms*, **13**, 1992, 33–54.

[18] M.T. Goodrich and R. Tamassia, "Dynamic Trees and Dynamic Point Location," *Proc. 23rd ACM Symp. on Theory of Computing*, 1991, 523–533.

[19] M.T. Goodrich and R. Tamassia, "Dynamic Ray Shooting and Shortest Paths via Balanced Geodesic Triangulations," *Proc. 9th ACM Symp. on Computational Geometry*, 1993, 318–327.

[20] L.J. Guibas, E.M. McCreight, M.F. Plass, and J.R. Roberts, "A New Representation for Linear Lists," *Proc. 9th ACM Symp. on Theory of Computing*, 1977, 49–60.

[21] L.J. Guibas and R. Sedgewick, "A Dichromatic Framework for Balanced Trees," *Proc. 19th IEEE Symp. on Foundations of Computer Science*, 1978, 8–21.

[22] S. Huddleston and K. Mehlhorn, "A New Data Structure for Representing Sorted Lists," *Acta Informatica*, **17**, 1982, 157–184.

[23] D. Kirkpatrick, "Optimal Search in Planar Subdivision," *SIAM Journal on Computing*, Vol. 12, No. 1, February 1983, pp. 28–35.

[24] D.E. Knuth, *Sorting and Searching*, Addison-Wesley, Reading, Mass., 1973.

[25] S.R. Kosaraju, "Localized Search in Sorted Lists," in *Proc. 13th Annual ACM Symp. on Theory of Computing*, 1981, pp. 62-69.

[26] S.R. Kosaraju, "An Optimal RAM Implementation of Catenable Min Double-ended Queues," *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, 1994, 195–203.

[27] H.T. Kung, F. Luccio, F.P. Preparata, "On Finding the Maxima of a Set of Vectors," *J. ACM*, Vol. 22, No. 4, 1975, pp. 469–476.

[28] D.T. Lee and F.P. Preparata, "Location of a Point in a Planar Subdivision and its Applications," *SIAM J. Computing*, Vol. 6, No. 3, 594–606, 1977.

[29] D.T. Lee and F.P. Preparata, "Computational Geometry—A Survey," *IEEE Trans. on Computers*, **C-33**(12), 1984, 872–1101.

[30] K. Mehlhorn, *Data Structures and Algorithms 1: Sorting and Searching*, Springer-Verlag, 1984.

[31] K. Mehlhorn, *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, Springer-Verlag, 1984.

[32] K. Mehlhorn and S. Näher, "Dynamic Fractional Cascading," *Algorithmica*, **5**, 1990, 215–241.

[33] I. Nievergelt and E.M. Reingold, "Binary Search Trees of Bounded Balance," *SIAM J. Comput.*, **2**, 1973, 33–43.

[34] M. Overmars, *The Design of Dynamic Data Structures*, Lecture Notes in Computer Science, Springer-Verlag, 1983.

[35] F.P. Preparata, "A New Approach to Planar Point Location," *SIAM J. Computing*, Vol. 10, No. 3, 1981, 73–83.

[36] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, NY, 1985.

[37] F.P. Preparata and R. Tamassia, "Fully Dynamic Point Location in a Monotone Subdivision," *SIAM J. Computing*, Vol. 18, No. 4, 811–830, 1989.

[38] Preparata, F.P. and R. Tamassia, "Dynamic Planar Point Location with Optimal Query Time," *Theoretical Computer Science*, Vol. 74, No. 1, 95–114, 1990.

[39] N. Sarnak and R.E. Tarjan, "Planar Point Location Using Persistent Search Trees," *Communications ACM*, Vol. 29, No. 7, 669–679, 1986.

[40] D.D. Sleator and R.E. Tarjan, "A Data Structure for Dynamic Trees," *J. Comput. and Sys. Sci.*, **26**, 362–391, 1983.

[41] R.E. Tarjan, *Data Structures and Network Algorithms*, SIAM, Philadelphia, PA, 1983.

[42] R.E. Tarjan, "Amortized computational complexity," *SIAM J. Alg. Disc. Meth.*, 6(2):306–318, April 1985.

An Algorithm for Approximate Closest-point Queries

Kenneth L. Clarkson
AT&T Bell Laboratories
Murray Hill, New Jersey 07974
e-mail: clarkson@research.att.com

Abstract

This paper gives an algorithm for approximately solving the *post office problem*: given n points (called sites) in d dimensions, build a data structure so that, given a query point q, a closest site to q can be found quickly. The algorithm is also given a relative error bound ϵ, and depends on a ratio ρ, which is no more than the ratio of the distance between the farthest pair of sites to the distance between the closest pair of sites. The algorithm builds a data structure of size $O(n\eta)O(1/\epsilon)^{(d-1)/2}$ in time $O(n^2\eta)O(1/\epsilon)^{(d-1)}$. Here $\eta = \log(\rho/\epsilon)$. With this data structure, a site is returned whose distance to a query point q is within $1+\epsilon$ of the distance of the closest site. A query needs $O(\log n)O(1/\epsilon)^{(d-1)/2}$ time, with high probability.

1 Introduction

The post-office problem is the following: given a set S of n points (called sites) in d dimensions, build a data structure so that given a query point q, the closest site to q can be found quickly. Many data structures have been proposed for this problem; their query times generally are faster than the trivial $O(n)$, and often are $O(\log n)$, as $n \to \infty$. The dependence of the query time on the dimension d is generally very steep for the nontrivial algorithms, at least $2^{\Omega(d)}$. Even heuristic algorithms that are fast in practice, such as bucketing and kd-trees, also have this exponential dependence. This is unfortunate, since many of the most interesting potential applications have d at least 10, and often much greater. Thus the promise of a query time of $O(\log n)$ is crushed by the "curse of dimensionality."

This paper gives a modest but significant improvement in the dimension dependence of one recent algorithm, that of Arya and Mount.[AM93] They attack the problem of *approximate* solution to the post-office problem; their procedure returns an *ϵ-closest* site, one whose distance to the query point q is within $1+\epsilon$ of closest. Here $\epsilon > 0$ is input when their data structure is built. The algorithm given here takes their approach to what is arguably its logical conclusion, reduces the query time from $\tilde{O}(\log^3 n)$ to $\tilde{O}(\log n)$,[1] and reduces the "constant" factors for the dimension from $O(1/\epsilon)^d$ to about $O(1/\epsilon)^{(d-1)/2}$. (The latter reduction applies to the storage and query-time bounds.) The new algorithm has an additional factor in storage and preprocessing time of $\log\rho$, where ρ is bounded above by the ratio of the distance between the sites farthest apart to the distance of the sites closest together. (In fact ρ is roughly the maximum, over all sites s, of the ratio of the distance to s to its farthest Delaunay neighbor, to the distance of s to its closest Delaunay neighbor. Hence $\log\rho$ is negligible relative to other factors.)

Arya *et al.* have more recently described a different approximation algorithm, based on quadtree techniques, that has better storage and preprocessing bounds than those given here, but with a query time that has a $\Omega(1/\epsilon)^d$ dependence.[AMN+94]

The new algorithm uses a technique previously used for polytope approximation [Cla93] in order to approximate a certain Voronoi region for each site by a simpler Voronoi region for the site. The approximation problem is solved using randomization. The algorithm also builds a data structure similar to a skip list, and uses randomization for that.[Pug90]

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

[1] Here $\tilde{O}(g(n))$ means $O(g(n))$ with probability at least $1 - 1/n^2$.

The following section first describes the general approach, and then develops that approach in the following subsections.

2 The algorithm

Following Arya and Mount, the general idea is to find, for each site s, a list of sites N_s with the following property: if s is not the closest site to the query point q, then there is a site in N_s closer to q than s. With this property, a simple search procedure will lead to the closest site: pick any site s; if a site $t \in N_s$ is closer to q, assign t to s and repeat; otherwise return s as closest.

This approach is not so interesting just yet. The list N_s must be the set of Delaunay neighbors of s, as the interested reader can easily show. This makes for a space requirement of $\Omega(n^2)$ in the worst case, for $d > 2$. Also, the query time is $\Omega(n)$ in the worst case: there is no speedup over the obvious algorithm. (For uniformly distributed points, the query time is more like $O(n^{1/d})$, so this approach is not entirely useless, however.)

For more interesting results, make the problem easier: instead of the closest site, find an ϵ-closest site. (Again, such a site has a distance to the query point that is within $1+\epsilon$ of the distance of the closest site's, for $\epsilon > 0$.) This is the approximate query problem solved by Arya and Mount. They used a collection of narrow cones to obtain their lists, in a way similar to Yao's use of them for finding minimum spanning trees[Yao82]. Here the approach is to go from the desired conditions on the lists to a problem similar to polytope approximation.

The modified construction begins as follows. For each site s, consider a list L_s with the following property: for any q, if there is $b \in S$ with

$$d(q,s) > (1+\epsilon)d(q,b),$$

then there is $b' \in L_s$ with

$$d(q,s) > (1+\epsilon')d(q,b'),$$

where $\epsilon' \equiv \epsilon/2$. (Note that if $\epsilon = 0$, then $L_s = N_s$.) Using such lists, the search procedure starts at any site s. If there is $t \in L_s$ with $d(q,s) > (1+\epsilon')d(q,t)$, then assign t to s and repeat. Otherwise, return s. With L_s as defined, the returned s is ϵ-closest. Note that the procedure makes progress at each step: the distance of the current site decreases by $1/(1+\epsilon')$.

Consider the condition satisfied by L_s in a contrapositive way. Fixing $s \in S$, let $\mathcal{N}_\epsilon(S)$ be defined by

$$\mathcal{N}_\epsilon(S) \equiv \{q \mid d(q,s) \leq (1+\epsilon)d(q,b) \text{ for all } b \in S\},$$

so that

$$\mathcal{N}_{\epsilon'}(L_s) = \{q \mid d(q,s) \leq (1+\epsilon')d(q,b) \text{ for all } b \in L_s\}.$$

The condition on L_s is equivalent to $\mathcal{N}_{\epsilon'}(L_s) \subset \mathcal{N}_\epsilon(S)$. The set $\mathcal{N}_\epsilon(S)$ is the Voronoi region of s in a certain multiplicatively weighted Voronoi diagram. The L_s we want is a small one that such that $\mathcal{N}_{\epsilon'}(L_s)$ is inside $\mathcal{N}_\epsilon(S)$; the problem of finding such an L_s can be solved by techniques previously applied to polytope approximation.[Cla93]

The following subsection discusses the problem of finding L_s; §2.2 bounds the size of such a list; §2.3 shows how to use the lists to obtain fast query times. Finally, §3 makes some concluding remarks.

2.1 Finding L_s

To find L_s, we'll translate the problem to $d+1$ dimensions using standard "lifting map" techniques.

If we put s at the origin, the region $\mathcal{N}_\epsilon(S)$ is the intersection of all regions of the form

$$\{z \mid z^2 \leq (1+\epsilon)^2(z-b)^2\},$$

where $b \in S$. The condition here is $z^2/(1+\epsilon)^2 \leq (z-b)^2$, or $\alpha z^2 \geq 2b \cdot z - b^2$, where $\alpha \equiv 1 - 1/(1+\epsilon)^2 \approx 2\epsilon$. Now let (z,y) denote a point in R^{d+1}, with $z \in R^d$ and $y \in R$. We have

$$\mathcal{N}_\epsilon(S) = \{z \mid \alpha y \geq 2b \cdot z - b^2 \text{ and } y = z^2\}.$$

So for $b \in S$, let $\mathcal{H}_{\epsilon,b}$ denote the halfspace

$$\mathcal{H}_{\epsilon,b} \equiv \{(z,y) \mid \alpha y \geq 2b \cdot z - b^2\},$$

and $\mathcal{P}_\epsilon(S)$ denote the polytope which is the intersection of such halfspaces,

$$\mathcal{P}_\epsilon(S) \equiv \bigcap_{b \in S} \mathcal{H}_{\epsilon,b}.$$

Then

$$\mathcal{N}_\epsilon(S) = \{z \mid (z,y) \in \mathcal{P}_\epsilon(S) \text{ and } y = z^2\}.$$

Let $\Psi \equiv \{(z,y) \mid y \geq z^2\}$. Then for $\mathcal{N}_{\epsilon'}(L_s) \subset \mathcal{N}_\epsilon(S)$, it is enough that

$$\mathcal{P}_{\epsilon'}(L_s) \cap \Psi \subset \mathcal{P}_\epsilon(S) \cap \Psi. \qquad (1)$$

The problem of finding L_s satisfying this condition can be solved using techniques as for polytope approximation[Cla93], which are very similar to techniques described long ago for linear programming[Cla88]. The general idea is this: give each site an integral weight, initially one. Take a random subset R of S, choosing each site with probability proportional to its weight, and making R about the right size. Test if R satisfies (1); if so, return it. Otherwise, we can derive from the testing of R a subset of S that we can expect to be small, and that must contain a site of an appropriate L_s. Double the weights of the sites in that subset. Repeat this process until done.

Here are a few more details of the algorithm: fix some optimal L satisfying (1), where the size c of C is as small as possible. Assume for the moment that c is known. (By using exponentially increasing estimates of c, we lose only a small factor in the size of the returned set, and in running time.) Give each $p \in S$ a weight, initially equal to one. Choose a random subset $R \subset S$ of size $cC_1 d \log c$, where C_1 is a small constant, choosing a site with probability proportional to its weight. For each $b \in S$, check that

$$\mathcal{P}_{\epsilon'}(R) \cap \Psi \subset \mathcal{H}_{\epsilon,b}. \qquad (2)$$

This is a convex programming problem, and as shown by Adler and Shamir, it is solvable in $O(n)$ expected time using a randomized procedure similar to one for linear programming.[AS90] (The base case, with $O(d^2)$ constraints, can be solved in polynomial time.[Vai89]) If all $b \in S$ satisfy (2), then return R as L_s. Suppose the condition fails for some $\hat{b} \in S$. Then there will be some vertex v of $\mathcal{P}_{\epsilon'}(R) \cap \Psi$ not in $\mathcal{H}_{\epsilon,\hat{b}}$; such a vertex can be returned as part of the output of the convex programming algorithm. Find all the halfspaces $\mathcal{H}_{\epsilon',b}$ that do not contain v, for $b \in S$. This set V of halfspaces will have relatively few members, with high probability, and will contain a member of the optimal set C. If the size of set V is less than $C_2 dn/r$, then double the weights of the corresponding sites. (Here C_2 is another small constant.) Repeat this procedure until a list L_s is returned.

An analysis of this procedure appears in [Cla93], and will appear in the full paper.

2.2 The size of L_s

This algorithm returns a set L_s of size within $O(d \log c)$ of the best possible size c, as can be shown as in [Cla93]. How large can c be? We have the following bound, which this subsection will prove.

Theorem 2.1 *The optimal size c of L_s is $O(1/\epsilon)^{(d-1)/2} d \log(\rho/\epsilon)$.*

We'll need the following lemma, due to Dudley.[Dud74]

Lemma 2.2 *Let $P \subset R^d$ be compact and convex, and contained in a ball of radius 1. For ϵ with $0 < \epsilon < 1$, there is a convex polytope $P' \supset P$ with $O(1/\epsilon)^{(d-1)/2}$ facets, and with P' within Haussdorf distance ϵ of P.*

We'll apply this lemma to bound the size of L_s. To do so, split Ψ into slabs

$$\Psi_i \equiv \{(z, y) \mid d_i \le y \le d_{i+1}\},$$

for $i = 0 \ldots m$, where $\sqrt{d_0} \equiv (1/2)\min_{b \in S} \|b\|$, and $d_i = 3d_{i-1}/2$, and m is large enough that $\sqrt{d_m} > (2/\alpha)\max_{b \in S} \|b\|$. We have $m = \log O(\rho/\alpha)$. The following lemma implies that only the points of these slabs need be considered.

Lemma 2.3 *Every halfspace $\mathcal{H}_{\epsilon,b}$ contains all points of Ψ not in some Ψ_i.*

Proof. We need to show that $y \ge z^2$ and either $y \ge d_m$ or $y \le d_0$ imply $\mathcal{H}_{\epsilon,b}$ for any $b \in S$. Since $b \cdot z$ is maximized for given $\|z\|$ when $z = b\|z\|/\|b\|$, when showing that $(z, y) \in \mathcal{H}_{\epsilon,b}$ we can assume $z = \gamma b$ for some γ. If $z^2 \ge d_m$, the minimum value of y to consider is z^2, so it's enough to show that $\alpha z^2 \ge 2b \cdot z - b^2$, or $\alpha\gamma^2 b^2 \ge 2\gamma b^2 - b^2$. This is implied by $\gamma \ge (2/\alpha)$. If $z^2 \le d_m$, then similarly we need $\alpha y \ge 4b^2/\alpha \ge 2\gamma b^2 - b^2$, which holds for $\gamma \le 2/\alpha$. When $y \le d_0$, we have $z^2 \le y \le b^2/4$, and so we need $\alpha z^2 \ge 2\gamma b^2 - b^2$, where $\gamma \le 1/2$. Since $\alpha z^2 \ge 0$, the result follows. □

Consider a given slab Ψ_i. We have

$$\alpha d_i \ge \frac{5}{4}\alpha' d_{i+1}, \qquad (3)$$

for sufficiently small ϵ. Consider the sets

$$\mathcal{P}^a \equiv \bigcap_{b \in S} \{(z, y) \mid \alpha' d_{i+1} \ge 2b \cdot z - b^2\},$$

and

$$\mathcal{P}^b \equiv \bigcap_{b \in S} \{(z, y) \mid \alpha d_i \ge 2b \cdot z - b^2\}.$$

Then (*considering only points in* Ψ_i)

$$\mathcal{P}_{\epsilon'}(S) \subset \mathcal{P}^a \subset \mathcal{P}^b \subset \mathcal{P}_{\epsilon}(S).$$

Moreover, condition (3) implies that there is a gap between $\mathcal{P}^a$ and $\mathcal{P}^b$ that we'll use shortly. Now let ball

$$B' \equiv \{z \mid z^2 \le d_{i+1}\},$$

so that all points in Ψ_i have z projections in B', and let

$$B \equiv \{z \mid z^2 \le (1 + 2\epsilon)^2 d_{i+1}\}.$$

Let $\mathcal{P}_p^a \equiv \{z \mid (z,y) \in \mathcal{P}^a\}$, and similarly define $\mathcal{P}_p^b$. Then $\mathcal{P}_p^a \cap B' \subset \mathcal{P}_p^b \cap B$, and by (3), every point of the former is at least

$$\frac{\alpha' d_{i+1}/4}{2\|b\|}$$

from any point not in the latter. If a plane $\mathcal{H}_{\epsilon',b}$ does not contain Ψ_i, and so is relevant here, then as in the proof of Lemma 2.3, we have $d_{i+1} \geq b^2/4$. This says that gap between $\mathcal{P}_p^a \cap B'$ and $\mathcal{P}_p^b \cap B$ is proportional to $\epsilon\sqrt{d_{i+1}}$, and so by appropriate scaling we can apply Lemma 2.2 to find a polytope $\mathcal{P}_p^c$ such that

$$\mathcal{P}_p^a \cap B' \subset \mathcal{P}_p^c \subset \mathcal{P}_p^b \cap B$$

and $\mathcal{P}^c$ has $O(1/\epsilon)^{(d-1)/2}$ facets. This implies that there is a polytope $\mathcal{P}^c$ in R^{d+1} of the same complexity, such that relative to Ψ_i,

$$\mathcal{P}_{\epsilon'}(S) \subset \mathcal{P}^c \subset \mathcal{P}_\epsilon(S).$$

It not hard to show that there is a "coarsening" polytope $\mathcal{P}_{\epsilon'}(L_s^i)$, with at most d times as many facets as $\mathcal{P}^c$, such that $\mathcal{P}_{\epsilon'}(L_s^i) \cap \Psi_i \subset \mathcal{P}_\epsilon(S)$.

By putting the lists L_s^i together into L_s, we obtain the size bound of the theorem of this subsection.

2.3 Solving closest-point problems

How can a fast query time be obtained using the lists L_s? Just as with Arya and Mount's work, a skip-list approach is helpful.[Pug90] Choose a family of subsets of S as follows: let $R_0 \equiv S$; to obtain R_{j+1} from R_j, pick each element of R_j to be in R_{j+1} with probability 1/2. Repeat this process until an empty R_k is obtained. If $s \in R_j$ but not R_{j+1}, say that s has level j. Construct the lists $L_{s,j}$ for each R_j, and so $s \in S$ has lists for each subset up to its level. To answer a query, start with some $s \in R_{k-1}$, and find the ϵ-closest site t_{k-1} in R_{k-1} using the lists $L_{s,k-1}$. Now find an ϵ-closest site t_{k-2} in R_{k-2}, starting with t_{k-1}. Repeat until t_0 is found, and return t_0 as an ϵ-closest site in S.

The correctness of this procedure should be clear. How much time does it take? Since each list is bounded in size by $O(dc \log c)$, where c is $O(1/\epsilon)^{(d-1)/2} d \log(\rho/\epsilon)$, the query time is equal to $O(dc \log c)$ times the number of sites visited in the procedure.

It is worthwhile to compare this procedure with one that finds the closest site in R_j at stage j, not just the ϵ-closest. Suppose we have t_j as the ϵ-closest at some stage, but indeed a site t is closest in R_j. When finding the ϵ-closest in R_{j+1}, the approximate procedure will in two steps find a site t' in R_{j+1} such that $d(q,t') \leq d(q,t)/(1+\epsilon')^2$. (Here we assume that the search in R_{j+1} takes at least two steps.) Since $(1+\epsilon')^2 \geq (1+\epsilon)$ for $\epsilon \geq 0$, we know that t' is closer to q than t. The number of sites visited at stage $j+1$ for the exact procedure is proportional to the number of sites of R_{j+1} closer to q than t; hence the number of sites visited for the approximate procedure in R_{j+1} is no more than 2 plus the number for the exact procedure.

To analyze the exact search procedure, we can follow Sen's analysis of skip lists.[Sen] Look at the search procedure "backwards": starting at the closest site to q in R_j, visit sites in R_j in order of increasing distance, until a site also in R_{j+1} is encountered. Call this a *level jump*. Once the level jump occurs, only sites in R_{j+1} are visited. The probability of a level jump at a given visited node is 1/2. Thus the probability that at least k level jumps occur in v node visits is the probability that a binomially distributed random variable has at least k successes in v trials. The query time can be greater than V only if either the number of level jumps exceeds K or if fewer than K level jumps occur in V attempts; the former probability is no more than $n/2^K$, which we'll need less than some probability P_1. This implies $K \geq \lg(n/P_1)$. The probability of fewer than K level jumps in V trials can be bounded using Chernoff bounds for the binomial; letting $\gamma \equiv 2K/V$, it is $\exp(-V(1-\gamma)^2/2)$. The probability that the query time exceeds $2\lg(n/P_1)/\gamma$ for a given point q is therefore at most $P_1 + \exp(-\lg(n/P_1)(1-\gamma)^2/\gamma)$. Hence, setting $P_1 = 1/n^Q$, an $O(Q)\log n$ query time is achievable with failure probability $O(1/n^Q)$.

This analysis applies only to a single given point q; what about arbitrary points? As with similar situations in randomized geometric algorithms, a good query time holds for all points because there are $n^{d^{O(1)}}$ combinatorially distinct classes of points. That is, in an exact search algorithm, two points q_1 and q_2 will have the same sequence of visited sites, and so the same query time, if the distance order on the sites induced by the two points is the same. In other words, whether we sort the sites in order of distance from q_1, or sort them in order of distance from q_2, we get the same sorted order. How many classes of points are distinct in this way? Let $\mathcal{B}$ be the set of $\binom{n}{2}$ perpendicular bisector hyperplanes of pairs of sites, and let $\mathcal{A}(\mathcal{B})$ be the subdivision of R^d induced by those bisectors. Then all points in one cell (block) of $\mathcal{A}(\mathcal{B})$ induce the same distance orders, and so have the same query time. The number of cells of $\mathcal{A}(\mathcal{B})$ is $\binom{\binom{n}{2}}{d} < n^{2d}$. Thus a query time for any point of

$O(\log n)$ occurs with probability $1 - 1/n^{\Omega(1)}$.

Queries can be made a bit faster by splitting up the each list $L_{s,j}$ into lists $L^i_{s,j}$, where the superscript corresponds to the slabs Ψ_i in §2.2. When searching for a given site s at a given stage j, the list $L^i_{s,j}$ with $i = 2 \lg d(s,q)$ can be used. This gives a query time independent of ρ.

3 Concluding Remarks

It should be possible to have an algorithm that is polynomial in d, by making ϵ a sufficiently large constant, as in Bern's note.[Ber93]

References

[AM93] S. Arya and D. M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 271–280, 1993.

[AMN+94] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and Angela Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, 1994.

[AS90] I. Adler and R. Shamir. A randomization scheme for speeding up algorithms for linear and convex quadratic programming problems with a high constraints-to-variables ratio. Technical Report 21-90, Rutgers Univ., May 1990. To appear in *Math. Programming*.

[Ber93] M. Bern. Approximate closest-point queries in high dimensions. *Inform. Process. Lett.*, 45:95–99, 1993.

[Cla88] K. L. Clarkson. A Las Vegas algorithm for linear programming when the dimension is small. In *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 452–456, 1988. Revised version: Las Vegas algorithms for linear and integer programming when the dimension is small (preprint).

[Cla93] K. L. Clarkson. Algorithms for polytope covering and approximation. In *Proc. 3rd Workshop Algorithms Data Struct.*, Lecture Notes in Computer Science, 1993.

[Dud74] R. M. Dudley. Metric entropy of some classes of sets with differentiable boundaries. *J. Approximation Theory*, 10:227–236, 1974.

[Pug90] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 35:668–676, 1990.

[Sen] S. Sen. Some oberservations on skip lists.

[Vai89] P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 338–343, 1989.

[Yao82] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11:721–736, 1982.

New techniques for exact and approximate dynamic closest-point problems

Sanjiv Kapoor[*] Michiel Smid[†]

Abstract

Let S be a set of n points in $\mathbb{R}^D$. It is shown that a range tree can be used to find an L_∞-nearest neighbor in S of any query point, in $O((\log n)^{D-1} \log\log n)$ time. This data structure has size $O(n(\log n)^{D-1})$ and an amortized update time of $O((\log n)^{D-1} \log\log n)$. This result is used to solve the $(1+\epsilon)$-approximate L_2-nearest neighbor problem within the same bounds. In this problem, for any query point p, a point $q \in S$ is computed such that the euclidean distance between p and q is at most $(1+\epsilon)$ times the euclidean distance between p and its true nearest neighbor. This is the first dynamic data structure for this problem having close to linear size and polylogarithmic query and update times.

New dynamic data structures are given that maintain a closest pair of S. For $D \geq 3$, a structure of size $O(n)$ is presented with amortized update time $O((\log n)^{D-1} \log\log n)$. For $D = 2$ and any non-negative integer constant k, structures of size $O(n \log n/(\log\log n)^k)$ (resp. $O(n)$) are presented having an amortized update time of $O(\log n \log\log n)$ (resp. $O((\log n)^2/(\log\log n)^k)$). Previously, no deterministic linear size data structure having polylogarithmic update time was known for this problem.

1 Introduction

Closest-point problems are among the basic problems in computational geometry. In such problems, a set S of n points in $\mathbb{R}^D$ is given and we have to store it in a data structure such that a point in S nearest to a query point can be computed efficiently, or we have to compute a closest pair in S, or for each point in S another point in S that is closest to it. Distances are measured in the L_t-metric, for a fixed t, $1 \leq t \leq \infty$. These problems are known as the *nearest-neighbor problem*, the *closest pair problem*, and the *all-nearest-neighbors-problem.* In the dynamic version, the set S is changed by insertions and deletions of points.

The planar version of the nearest-neighbor problem can be solved optimally, i.e., with $O(\log n)$ query time using $O(n)$ space, by means of Voronoi diagrams. In higher dimensions, however, the situation is much worse. The best results known are due to Clarkson [3] and Arya et al.[1]. In [3], a randomized data structure is given that finds a nearest-neighbor of a query point in $O(\log n)$ expected time. This structure has size $O(n^{\lceil D/2 \rceil + \delta})$, where δ is an arbitrarily small positive constant. In [1], the problem is solved with an expected query time of $O(n^{1-1/\lceil (D+1)/2 \rceil} (\log n)^{O(1)})$ using $O(n \log\log n)$ space. It seems that in higher dimensions it is impossible to obtain polylogarithmic query time using $O(n(\log n)^{O(1)})$ space. Moreover, even in the planar case there is no dynamic data

[*] Department of Computer Science, Indian Institute of Technology, Hauz Khas, New Delhi 110016, India. E-mail: skapoor@cse.iitd.ernet.in

[†] Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany. E-mail: michiel@mpi-sb.mpg.de. This author was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

structure known that has polylogarithmic query and update times and that uses $O(n(\log n)^{O(1)})$ space.

Therefore, it is natural to ask whether the *approximate nearest-neighbor problem* allows more efficient solutions. Let $\epsilon > 0$. A point $q \in S$ is called a $(1+\epsilon)$-approximate neighbor of a point $p \in \mathbb{R}^D$, if $d_t(p,q) \leq (1+\epsilon)d_t(p,p^*)$, where $p^* \in S$ is the true nearest-neighbor of p.

This approximate neighbor problem was considered in Arya et al.[1, 2]. In the latter paper, the problem is solved optimally: They give a deterministic data structure of size $O(n)$ that can find a $(1+\epsilon)$-approximate neighbor in $O(\log n)$ time, for any positive constant ϵ. At this moment, however, no dynamic data structures are known for this problem. In this paper, we prove the following results.

Theorem 1 *Let S be a set of n points in $\mathbb{R}^D$. There exists a dynamic data structure of size $O(n(\log n)^{D-1})$ that, given a query point $p \in \mathbb{R}^D$, finds an L_∞-neighbor of p in S in $O((\log n)^{D-1}\log\log n)$ time. This data structure has an amortized update time of $O((\log n)^{D-1}\log\log n)$.*

Theorem 2 *Let S be a set of n points in $\mathbb{R}^D$ and let ϵ be a positive constant. There exists a dynamic data structure of size $O(n(\log n)^{D-1})$ that, given a query point $p \in \mathbb{R}^D$, finds a $(1+\epsilon)$-approximate L_2-neighbor of p in S in $O((\log n)^{D-1}\log\log n)$ time. This data structure has an amortized update time of $O((\log n)^{D-1}\log\log n)$.*

We also consider the dynamic closest pair problem, in which we have to maintain a closest pair under insertions and deletions. This problem has been investigated only recently. In Table 1, we give an overview of the currently best known data structures for maintaining a closest pair under insertions and/or deletions of points. For an up-to-date survey, we refer the reader to Schwarz's Ph.D. Thesis [9].

Note that all data structures of Table 1 are deterministic and can be implemented in the algebraic computation tree model. If we add randomization to this model, then there is a data structure of size $O(n)$ that maintains a closest pair in $O((\log n)^2)$ expected time per update. (See Golin et al.[5].)

In this paper, we give new deterministic data structures for the dynamic closest pair problem. These structures are based on our solution to the L_∞-neighbor problem, on data structures for maintaining boxes of constant overlap, and on a new transformation. Given *any* dynamic closest pair data structure having more than linear size, this transformation produces another dynamic closest pair structure that uses less space. The complexity bounds of the new structures are shown in the last three lines of Table 1. The results of the last two lines are obtained by applying the transformation repeatedly. Note that we obtain the first linear size deterministic data structure that maintains a closest pair in polylogarithmic time. As a by-product, we prove:

Theorem 3 *Let S be a set of n points in $\mathbb{R}^D$. There exists a dynamic data structure of size $O(n(\log n)^{D-1})$ that maintains an L_∞-neighbor of each point in S. This data structure has an amortized update time of $O((\log n)^{D-1}\log\log n)$.*

All our algorithms use classical and well-understood data structuring techniques, such as range trees, segment trees and dynamic fractional cascading.

The rest of this paper is organized as follows. In Section 2, we show how range trees can be used to solve the L_∞-neighbor problem. We also give the data structure for solving the approximate L_2-nearest neighbor problem.

Our first data structure for maintaining the closest pair stores a dynamically changing set of boxes that are of constant overlap, i.e., there is a constant c such that each box contains the centers of at most c boxes in its interior. We have to maintain such boxes under insertions and deletions such that for any query point $p \in \mathbb{R}^D$, we can find all boxes that contain p. In Section 3, we give two data structures for this problem. The first one is based on segment trees. Therefore, its size is superlinear. We also give a linear size solution having a slightly worse update time in the planar case. This solution is based on skewer trees. (See [4, 11].)

In Section 4, we use the results obtained to give a new data structure that maintains the closest pair in a point set. This structure has size $O(n(\log n)^{D-1})$ and an amortized update time of

mode	dimension	update time	w/a	space	reference
insertions	$D \geq 2$	$\log n$	w	n	[9, 10]
deletions	$D \geq 2$	$(\log n)^D$	a	$n(\log n)^{D-1}$	[13]
fully dynamic	$D \geq 2$	$\sqrt{n} \log n$	w	n	[8]
fully dynamic	$D \geq 2$	$(\log n)^D \log\log n$	a	$n(\log n)^D$	[12]
fully dynamic	$D \geq 3$	$(\log n)^{D-1} \log\log n$	a	n	this paper
fully dynamic	2	$\log n \log\log n$	a	$n \log n/(\log\log n)^k$	this paper
fully dynamic	2	$(\log n)^2/(\log\log n)^k$	a	n	this paper

Table 1: Deterministic data structures for the dynamic closest pair problem. Distances are measured in the L_t-metric, for a fixed $1 \leq t \leq \infty$. In the last two lines, k is an arbitrary non-negative integer constant. The update times are either worst-case (w) or amortized (a).

$O((\log n)^{D-1} \log\log n)$. It immediately gives the dynamic data structure for the all-L_∞-nearest-neighbors problem. In Section 5, we give the transformation that reduces the size of a dynamic closest pair structure. Applying this transformation repeatedly to the structure of Section 4 gives the new results mentioned in Table 1.

2 The L_∞-neighbor problem

Let S be a set of n points in $\mathbb{R}^D$. We want to store this set into a data structure such that for any query point $p \in \mathbb{R}^D$, we can find a point in S having minimal L_∞-distance to p. Such a point is called an *L_∞-neighbor* of p in S.

Let q be an L_∞-neighbor of p in the set $\{s \in S : s_1 \geq p_1\}$. We call q a *right-L_∞-neighbor* of p in S. Similarly, a point r is called a *left-L_∞-neighbor* of p in S, if it is an L_∞-neighbor of p in the set $\{s \in S : s_1 \leq p_1\}$. In order to guarantee that both neighbors always exist, we add the $2D$ artificial points $(a_1, \ldots, a_D)$, where all a_i are zero, except for one which is either ∞ or $-\infty$, to the set S.

The data structure that solves the L_∞-neighbor problem is just a D-dimensional *range tree* storing the points of S and the artificial points. (See [7].) We recall an important property of range trees: Let p be any point in $\mathbb{R}^D$. Consider the set $\{r \in S : r_1 \geq p_1\}$, i.e., the set of all points in S having a first coordinate that is at least equal to p's first coordinate. Using the range tree, we can decompose this set into $O(\log n)$ canonical subsets: Initialize $M := \emptyset$. Starting in the root of the main tree, search for the leftmost leaf storing a point whose first coordinate is at least equal to p_1. During this search, each time we move from a node v to its left son, add the right son of v to the set M. Let v be the leaf in which this search ends. If the point stored in this leaf has a first coordinate that is at least equal to p_1, then add v to the set M. It is well known that the final set M satisfies $\{r \in S : r_1 \geq p_1\} = \bigcup_{v \in M} S_v$. Here, S_v denotes the set of points that are stored in the subtree of v. Range trees can be maintained under insertions and deletions of points. The update algorithms use dynamic fractional cascading. (See [6].) A D-dimensional range tree for a set of n points has $O(n(\log n)^{D-1})$ size and can be maintained in $O((\log n)^{D-1} \log\log n)$ amortized time per insertion and deletion.

We use the following notations. For any point $p = (p_1, p_2, \ldots, p_D) \in \mathbb{R}^D$, we denote by p' the point $(p_2, \ldots, p_D)$ in $\mathbb{R}^{D-1}$. If S is a set of points in $\mathbb{R}^D$, then $S' = \{p' : p \in S\}$. In Figure 1, our recursive algorithm is given that finds an L_∞-neighbor of any query point $p \in \mathbb{R}^D$.

Lemma 1 *1. If the variable stop has value false after the while-loop of Stage 2 has been completed, then the set C contains a right-L_∞-neighbor of p in S.*

2. If the variable stop has value true after the while-loop of Stage 2 has been completed, then the set $C \cup S_v$ contains a right-L_∞-neighbor of p in S.

3. During the while-loop of Stage 3, the set $C \cup S_v$ contains a right-L_∞-neighbor of p in S.

```
Algorithm Neighbor($p, S, D$)    (* returns an $L_\infty$-neighbor of $p$ in $S$ *)
begin
1. Assume $D = 1$.
   $q^L :=$ maximal element in the 1-dimensional range tree that is less than $p$;
   $q^R :=$ minimal element in the 1-dimensional range tree that is at least equal to $p$;
   if $|p - q^R| \leq |p - q^L|$ then return $q^R$ else return $q^L$ fi;
2. Assume $D \geq 2$.
2a. Compute a right-$L_\infty$-neighbor:
    Stage 1. Compute the set $M$ of nodes of the main tree, see above.
             Number these nodes $v_1, v_2, \ldots, v_m$, $m = |M|$, where $v_i$ is closer to
             the root than $v_{i-1}$, $2 \leq i \leq m$.
    Stage 2. (* one of the sets $S_{v_i}$ contains a right-$L_\infty$-neighbor of $p$ *)
             $C := \emptyset$; $i := 1$; $stop := false$;
             while $i \leq m$ and $stop = false$
             do $x' :=$ Neighbor($p', S'_{v_i}, D - 1$);
                $r :=$ the point stored in the rightmost leaf of the subtree of $v_i$;
                if $d_\infty(p', x') > |p_1 - r_1|$
                then $C := C \cup \{x\}$; $i := i + 1$
                else $v := v_i$; $stop := true$
                fi
             od;
             if $stop = false$
             then $q^R :=$ a point of $C$ having minimal $L_\infty$-distance to $p$;
                  goto 2b
             fi;
    Stage 3. (* the set $C \cup S_v$ contains a right-$L_\infty$-neighbor of $p$ *)
             while $v$ is not a leaf
             do $w :=$ left son of $v$;
                $x' :=$ Neighbor($p', S'_w, D - 1$);
                $r :=$ the point stored in the rightmost leaf of the subtree of $w$;
                if $d_\infty(p', x') > |p_1 - r_1|$
                then $C := C \cup \{x\}$; $v :=$ right son of $v$
                else $v := w$
                fi
             od;
             $q^R :=$ a point of $C \cup S_v$ having minimal $L_\infty$-distance to $p$;
2b. Compute a left-$L_\infty$-neighbor:
    In a completely symmetric way, compute a left-$L_\infty$-neighbor $q^L$ of $p$;
2c. if $d_\infty(p, q^R) \leq d_\infty(p, q^L)$ then return $q^R$ else return $q^L$ fi
end
```

Figure 1: Finding an L_∞-neighbor.

This lemma implies that point q^R which is computed in part **2a** is a right-L_∞-neighbor of p in S. Similarly, point q^L computed in part **2b** is a left-L_∞-neighbor of p. This proves that the point that is returned in part **2c** is an L_∞-neighbor of p.

Algorithm *Neighbor(p, S, D)* can be implemented using fractional cascading, such that its running time is bounded by $O((\log n)^{D-1})$. Hence, we have a data structure for the L_∞-neighbor problem that has a query time of $O((\log n)^{D-1})$ and that uses $O(n(\log n)^{D-1})$ space. Range trees can be maintained under insertions and deletions of points. Because of dynamic fractional cascading, the query time increases by a factor of $O(\log\log n)$. This proves Theorem 1.

Let S be a set of n points in $\mathbb{R}^D$ and let $\epsilon > 0$ be a fixed constant. For any point $p \in \mathbb{R}^D$, we denote by p^* its L_2-neighbor in S. A point $q \in S$ is called a $(1+\epsilon)$-*approximate* L_2-*neighbor* of p if $d_2(p,q) \leq (1+\epsilon)d_2(p,p^*)$. We want to store the set S in a data structure such that for any query point $p \in \mathbb{R}^D$ we can find a $(1+\epsilon)$-*approximate* L_2-*neighbor* of it.

Let $p \in \mathbb{R}^D$ and let q be an L_∞-neighbor of p in S. It is easy to show that q is a $\sqrt{D}$-approximate L_2-neighbor of p. We extend this as follows. Let $(\mathcal{F}_i)$ be a family of orthonormal coordinate systems all sharing the same origin. Assume that for any point x in $\mathbb{R}^D$ there is an index i such that in $\mathcal{F}_i$ all coordinates of x are almost equal, i.e., within $1+\epsilon$ from each other. In Yao [15], it is shown how such a family consisting of $O((c/\epsilon)^{D-1})$ coordinate systems can be constructed. In the planar case, this family is obtained by rotating the X- and Y-axes over an angle of $i \cdot \phi$, $0 \leq i < 2\pi/\phi$, where $0 < \phi < \pi/4$ is such that $\tan\phi = \epsilon/(2+\epsilon)$.

The data structure for approximate L_2-neighbor queries: For each index i, let S_i denote the set of points in S with coordinates in the system $\mathcal{F}_i$. The data structure consists of a collection of range trees; the i-th range tree stores the set S_i.

Finding an approximate L_2-neighbor: Let $p \in \mathbb{R}^D$ be a query point. For each index i, use the i-th range tree to find an L_∞-neighbor $q^{(i)}$ of p in S_i. Report an L_∞-neighbor that has minimal L_2-distance to p.

It can be shown that this query algorithm reports a $(1+\epsilon)$-approximate L_2-neighbor of p. This proves Theorem 2.

3 The containment problem for boxes of constant overlap

A *box* is a D-dimensional axes-parallel cube, i.e., it is of the form $[a_1 : a_1 + \delta] \times [a_2 : a_2 + \delta] \times \ldots \times [a_D : a_D + \delta]$, for real numbers $a_1, a_2, \ldots, a_D$ and $\delta > 0$. The *center* of this box is the point $(a_1 + \delta/2, a_2 + \delta/2, \ldots, a_D + \delta/2)$.

Let S be a set of n boxes in $\mathbb{R}^D$ that are of *constant overlap*, i.e., there is an integer constant c_D such that each box B of S contains the centers of at most c_D boxes in its interior. (Here, we also count the center of B itself. Note that many boxes may have their center on the boundary of B.)

We want to store these boxes in a data structure such that for any query point $p \in \mathbb{R}^D$, we can find all boxes that contain p.

Remark: Note that we distinguish between "being contained in a box" and "being contained in the interior of a box".

In the full paper it is shown that any point $p \in \mathbb{R}^D$ is contained in at most $2^{1+D^2}c_D$ boxes of S. (The proof is technical.)

3.1 A solution based on segment trees

We start with the one-dimensional case. Let S be a set of n intervals $[a_j : b_j]$, $1 \leq j \leq n$, that are of constant overlap with constant c. We store the intervals in the leaves of a balanced binary search tree T, sorted by their right endpoints. The leaves of this tree are threaded in a doubly-linked list. Let $p \in \mathbb{R}$ be a query element. Search in T for the leftmost leaf containing a right endpoint that is at least equal to p. Starting in this leaf, walk along the leaves to the right and report all intervals encountered that contain p. Stop walking as soon as $4c$ intervals have been reported, or c intervals have been encountered that do not contain p. It is clear that this data structure solves the one-dimensional problem in $O(n)$ space with a query time of $O(\log n)$. Moreover, intervals can be inserted and deleted in $O(\log n)$ time.

We next consider the D-dimensional case, where $D \geq 2$. Let S be a set of n boxes in $\mathbb{R}^D$ that are of constant overlap with constant c_D. If $B = B_1 \times B_2 \times \ldots \times B_D$ is a box in $\mathbb{R}^D$, then B' denotes the box $B_2 \times \ldots \times B_D$ in $\mathbb{R}^{D-1}$. Similarly, S' denotes the set $\{B' : B \in S\}$. Recall that we use a similar notation for points.

The D-dimensional structure for $D \geq 2$: Let $a_1 < a_2 < \ldots < a_m$, where $m \leq 2n$, be the sorted sequence of all distinct endpoints of the intervals of the first coordinates of the boxes in S. We store the *elementary intervals* $(-\infty : a_1), [a_1 : a_1], (a_1 : a_2), [a_2 : a_2], \ldots, (a_{m-1} : a_m), [a_m : a_m], (a_m : \infty)$ in this order in the leaves of a balanced binary search tree, called the *main tree.* Each node v of this tree has associated with it an interval I_v being the union of the elementary intervals of the leaves in the subtree of v. Let S_v be the set of all boxes $B = B_1 \times B_2 \times \ldots \times B_D$ in S such that $I_v \subseteq B_1$ and $I_{f(v)} \not\subseteq B_1$. ($f(v)$ is the father of v.)

Each node v of the main tree contains (a pointer to) an *associated structure* for the set S_v: Partition this set into S_{vl}, S_{vc} and S_{vr}, consisting of those boxes of S_v whose centers lie to the left of the "vertical" slab $I_v \times \mathbb{R}^{D-1}$, in or on the boundary of this slab, and to the right of this slab, respectively.

The associated structure of v consists of three recursively defined $(D-1)$-dimensional structures for the sets S'_{vl}, S'_{vc} and S'_{vr}.

The query algorithm: Let $p = (p_1, p_2, \ldots, p_D) \in \mathbb{R}^D$ be a query point. Search in the main tree for the elementary interval that contains p_1. For each node v on the search path, recursively perform a $(D-1)$-dimensional query with the point p' in the three structures that are stored with v.

At the last level of the recursion, a one-dimensional query is performed in a binary search tree. In this tree, the algorithm stops if it has reported $2^{1+D^2} c_D$ boxes of S that contain p, or if it has encountered c_D boxes of S that do not contain p.

Lemma 2 *The query algorithm is correct.*

Proof: It is well known that the set of all boxes $B = B_1 \times \ldots \times B_D$ such that $p_1 \in B_1$ is exactly the union of all sets S_v, where v is a node on the search path to p_1. Hence, we only have to consider the nodes on this search path.

Let v be a node on the path to p_1. We have to find all boxes in S_v whose last $D-1$ intervals contain $(p_2, \ldots, p_D)$. We claim that the recursive queries in the three structures stored with v find these boxes. We know that there are at most $2^{1+D^2} c_D$ boxes that contain p. Hence, at the last level of the recursion, the query algorithm can stop as soon as it has reported this many boxes. It remains to prove that, at the last level, the query algorithm can stop if it has encountered c_D boxes that do not contain p.

Consider such a last level. That is, let $v_1, v_2, \ldots, v_{D-1}$ be nodes such that (1) $v_1 = v$, (2) v_i is a node of the main tree of one of the three structures that are stored with v_{i-1}, and (3) v_i lies on the search path to p_i.

The algorithm makes one-dimensional queries with p_D in the three structures—binary search trees—that are stored with v_{D-1}. Consider one such query. The algorithm searches for p_D. Let r be the leaf in which this search ends. Starting in r, the algorithm walks along the leaves to the right. During this walk, it encounters boxes that do or do not contain p. Assume that in leaf s, the c_D-th box is encountered that does not contain p. We have to show that the algorithm can stop in s. That is, we must show that all leaves to the right of s store boxes that do not contain p.

Assume this is not the case. Then, there is a box $A = [a_1 : a_1 + \alpha] \times [a_2 : a_2 + \alpha] \times \ldots \times [a_D : a_D + \alpha]$ that is stored in a leaf to the right of s and that contains p. Let $B^{(j)} = [b_{j1} : b_{j1} + \beta_j] \times [b_{j2} : b_{j2} + \beta_j] \times \ldots \times [b_{jD} : b_{jD} + \beta_j]$, $1 \leq j \leq c_D$, be the encountered boxes between r and s that do not contain p. In the full paper, we prove that A contains the centers of all these boxes in its interior. This is a contradiction. ■

Theorem 4 *Let S be a set of n boxes in $\mathbb{R}^D$ of constant overlap. There exists a dynamic data structure of size $O(n(\log n)^{D-1})$ such that for any point $p \in \mathbb{R}^D$, we can find all boxes of S that contain p in $O((\log n)^{D-1} \log\log n)$ time. This data structure has an amortized update time of $O((\log n)^{D-1} \log\log n)$.*

3.2 A solution based on skewer trees

In this section, we give a linear space solution to the box containment problem. This solution is based on skewer trees, introduced by Edelsbrunner et al. [4].

Let S be a set of n boxes in $\mathbb{R}^D$ that are of constant overlap with constant c_D. For $D = 1$, the data structure is the same as in the previous subsection.

The D-dimensional structure for $D \geq 2$: If S is empty, then the data structure is also empty. Assume that S is non-empty. Let γ be the median of the set $\{(a_1+b_1)/2 : [a_1 : b_1]\times[a_2 : b_2]\times\ldots\times[a_D : b_D] \in S\}$, and let ϵ be the largest element that is less than γ in the set of all elements a_1, $(a_1+b_1)/2$ and b_1, where $[a_1 : b_1] \times \ldots \times [a_D : b_D]$ ranges over S. Let $\gamma_1 := \gamma - \epsilon/2$ and let σ be the hyperplane in $\mathbb{R}^D$ with equation $x_1 = \gamma_1$.

Let $S_<$, S_0 and $S_>$ be the set of boxes $[a_1 : b_1] \times \ldots \times [a_D : b_D]$ in S such that $b_1 < \gamma_1$, $a_1 \leq \gamma_1 \leq b_1$ and $\gamma_1 < a_1$, respectively. The D-dimensional data structure for the set S is an augmented binary search tree—called the *main tree*—having the following form:

1. The root contains the hyperplane σ.

2. The root contains pointers to its left and right sons, which are D-dimensional structures for the sets $S_<$ and $S_>$, respectively.

3. The root contains (a pointer to) an *associated structure* for the set S_0: Partition this set into S_{0l} and S_{0r}, consisting of those boxes in S_0 whose centers lie to the left of the hyperplane σ, and to the right of σ, respectively. The associated structure of the root consists of two $(D-1)$-dimensional structures for the sets S'_{0l} and S'_{0r}.

The query algorithm and its analysis is similar to that of the previous subsection. The details can be found in the full paper. (See also Smid [11].)

Theorem 5 *Let S be a set of n boxes in $\mathbb{R}^D$ of constant overlap. There exists a data structure of size $O(n)$ such that for any point $p \in \mathbb{R}^D$, we can find all boxes of S that contain p in $O((\log n)^{D-1})$ time. This static data structure can be built in $O(n \log n)$ time.*

For the dynamic version of the problem, the query time is $O((\log n)^{D-1} \log\log n)$ and the size of the structure is $O(n)$. Boxes can be inserted and deleted in amortized time $O((\log n)^2 \log\log n)$.

4 Maintaining the closest pair

In this section, we apply the results obtained so far to maintain a closest pair of a point set under insertions and deletions. Let S be a set of n points in $\mathbb{R}^D$ and let $1 \leq t \leq \infty$ be a real number. We denote the L_t-distance between any two points p and q in $\mathbb{R}^D$ by $d(p,q)$. The pair $P, Q \in S$ is called a *closest pair* of S if $d(P,Q) = \min\{d(p,q) : p, q \in S, p \neq q\}$.

We introduce the following notations. For any point $p \in \mathbb{R}^D$, $box(p)$ denotes the smallest box centered at p that contains at least $(2D+2)^D$ points of $S \setminus \{p\}$. In other words, the side length of $box(p)$ is twice the L_∞-distance between p and its $(1 + (2D+2)^D)$-th resp. $(2D+2)^D$-th L_∞-neighbor, if $p \in S$ resp. $p \notin S$.

Let $N(p)$ be the set of points of $S \setminus \{p\}$ that are contained in the interior of $box(p)$. Note that $N(p)$ has size less than $(2D+2)^D$. In fact, $N(p)$ may even be empty.

Lemma 3 *The set $\{(p,q) : p \in S, q \in N(p)\}$ contains a closest pair of S.*

Proof: Let (P,Q) be a closest pair of S. We have to show that $Q \in N(P)$. Assume this is not the case. Let δ be the side length of $box(P)$. Since Q lies outside or on the boundary of this box, we have $d(P,Q) \geq \delta/2$.

Partition $box(P)$ into $(2D+2)^D$ subboxes with sides of length $\delta/(2D+2)$. Since $box(P)$ contains at least $1 + (2D+2)^D$ points of S, one of these subboxes contains at least two points. These two points have distance at most $D \cdot \delta/(2D+2) < \delta/2$, which is a contradiction, because (P,Q) is a closest pair of S. ∎

The set $\{box(p) : p \in S\}$ is of constant overlap: each box contains the centers of at most $(2D+2)^D$ boxes in its interior. These centers are precisely the points of $N(p) \cup \{p\}$. This fact and the above lemma suggest the following data structure.

The closest pair data structure:

1. The points of S are stored in a range tree.

2. The distances of the multiset $\{d(p,q) : p \in S, q \in N(p)\}$ are stored in a heap. With each distance, we store the corresponding pair of points. (Note that both $d(p,q)$ and $d(q,p)$ may occur in the heap.)

3. The points of S are stored in a dictionary. With each point p, we store a list containing the elements of $N(p)$. For convenience, we also call this list $N(p)$. With each point q in $N(p)$, we store a pointer to the occurrence of $d(p,q)$ in the heap.

4. The set $\{box(p) : p \in S\}$ is stored in the dynamic data structure of Theorem 4. This structure is called the *box tree*.

It follows from Lemma 3 that the pair of points that is stored with the minimal element of the heap is a closest pair of S. The update algorithms are rather straightforward. We only give the insertion algorithm.

The insertion algorithm: Let $p \in \mathbb{R}^D$ be the point to be inserted. Assume $p \notin S$.

1. Using the range tree, find the $(2D+2)^D$ L_∞-neighbors of p in S. The point among these neighbors having maximal L_∞-distance to p determines $box(p)$. The neighbors that are contained in the interior of $box(p)$ form the list $N(p)$.

2. Insert p into the range tree and insert the distances $d(p,q)$, $q \in N(p)$ into the heap. Then, insert p—together with the list $N(p)$—into the dictionary. With each point q in $N(p)$, store a pointer to $d(p,q)$ in the heap. Finally, insert $box(p)$ into the box tree.

3. Using the box tree, find all boxes that contain p. For each reported element $box(q)$, $q \neq p$, that contains p in its interior, do the following:

3.a. Search in the dictionary for q. Insert p into $N(q)$, insert $d(q,p)$ into the heap and store with p a pointer to $d(q,p)$.

3.b. If $N(q)$ has size less than $(2D+2)^D$, then the insertion algorithm is completed. Otherwise, if $N(q)$ has size $(2D+2)^D$, let $r_1, \ldots, r_l$ be all points in $N(q)$ that have maximal L_∞-distance to q. For each $1 \leq i \leq l$, delete r_i from $N(q)$ and delete $d(q,r_i)$ from the heap. Finally, delete $box(q)$ from the box tree and insert the box centered at q having r_1 on its boundary as the new $box(q)$.

During the update algorithms, we perform a constant number of query and update operations in the range tree, the box tree, the heap and the dictionary. Therefore, Theorems 1 and 4 imply:

Theorem 6 *Let S be a set of n points in $\mathbb{R}^D$ and let $1 \leq t \leq \infty$. There exists a data structure of size $O(n(\log n)^{D-1})$ that maintains an L_t-closest pair of S in $O((\log n)^{D-1} \log\log n)$ amortized time per insertion and deletion.*

Consider again our data structure. The box $box(p)$ that is associated with point p contains an L_∞-neighbor of p in $S \setminus \{p\}$. Therefore, the data structure can easily be adapted such that it maintains for each point in S its L_∞-neighbor. This proves Theorem 3.

5 A transformation for reducing the space complexity

The closest pair data structure of the previous section uses more than linear space. This raises the question if the same update time can be obtained using only linear space. Note that we have a linear space solution for maintaining the set $\{box(p) : p \in S\}$. (See Theorem 5.) For the L_∞-neighbor problem, however, no linear space solution having polylogarithmic query and update times is known. Hence, in order to reduce the space complexity, we should avoid using the range tree.

We will give a transformation that, given *any* dynamic closest pair data structure having more than linear size, produces another dynamic closest pair structure that uses less space.

Let DS be a data structure that maintains a closest pair in a set of n points in $\mathbb{R}^D$ under insertions and deletions. Let $S(n)$ and $U(n)$ denote the size and update time of DS, respectively. We assume that $S(n)$ and $U(n)$ are non-decreasing. Let $f(n)$ be a non-decreasing integer function such that $1 \leq f(n) \leq n/2$.

Let $S \subseteq \mathbb{R}^D$ be the current set of points. The cardinality of S is denoted by n. Our transformed data structure will be completely rebuilt after a sufficiently long sequence of updates. Let S_0 be the set of points at the moment of the most recent rebuilding and let n_0 be its size at that moment.

As in the previous section, for each $p \in \mathbb{R}^D$, $box(p)$ denotes the smallest box centered at p that contains at least $(2D+2)^D$ points of $S \setminus \{p\}$. The set of all points of $S \setminus \{p\}$ that are in the interior of this box is denoted by $N(p)$. If $p \in S_0$, then $box_0(p)$ denotes the smallest box centered at p that contains at least $(2D+2)^D$ points of $S_0 \setminus \{p\}$.

The transformed closest pair data structure:

1. The set S is partitioned into sets A and B such that $A \subseteq \{p \in S : p \in S_0 \wedge box(p) \subseteq box_0(p)\}$.

2. The distances of the multiset $\{d(p,q) : p \in A, q \in N(p)\}$ are stored in a heap. With each distance, we store the corresponding pair of points.

3. The boxes of the set $\{box_0(p) : p \in S_0\}$ are stored in a list, called the *box list*. With each element $box_0(p)$ in this list, we store a bit having value *true* if and only if $p \in A$. Moreover, if $p \in A$, we store with $box_0(p)$ the box $box(p)$.

4. The boxes of the set $\{box_0(p) : p \in S_0\}$ are stored in the *static* data structure of Theorem 5. This structure is called the *box tree.* With each box in this structure, we store a pointer to its occurrence in the box list.

5. The points of S are stored in a dictionary. With each point p, we store a bit indicating whether p belongs to A or B. If $p \in A$, then we store with p

5.a. a pointer to the occurrence of $box_0(p)$ in the box list, and

5.b. a list containing the elements of $N(p)$. For convenience, we also call this list $N(p)$. With each point q in $N(p)$, we store a pointer to the occurrence of $d(p,q)$ in the heap.

6. The set B is stored in the dynamic data structure DS, called the *B-structure.*

Lemma 4 *Let δ be the minimal distance stored in the heap and let δ' be the distance of a closest pair in B. Then,* $\min(\delta, \delta')$ *is the distance of a closest pair in the set S.*

Proof: Let (P,Q) be a closest pair in S. We distinguish two cases.

Case 1: At least one of P and Q is contained in A. Assume that $P \in A$. Since $box(P)$ contains at least $1 + (2D+2)^D$ points of S, Q is contained in the interior of this box. Hence, $Q \in N(P)$ and the distance $d(P,Q)$ is stored in the heap. Therefore, $\delta = d(P,Q)$. Clearly, $d(P,Q) \leq \delta'$. This proves that $d(P,Q) = \min(\delta, \delta')$.

Case 2: Both P and Q are contained in B. Then $\delta' = d(P,Q)$ and $d(P,Q) \leq \delta$. ■

Initialization: At the moment of initialization, $S = S_0 = A$ and $B = \emptyset$. Using Vaidya's algorithm [14], compute for each point p in S its $1 + (2D+2)^D$ L_∞-neighbors. The point among these neighbors having maximal L_∞-distance to p determines $box(p) = box_0(p)$. The neighbors (except p itself) that are contained in the interior of this box form the list $N(p)$. It is clear how the rest of the data structure can be built.

The insertion algorithm: Let $p \in \mathbb{R}^D$ be the point to be inserted.

1. Insert p into the dictionary and store with p a bit saying that p belongs to B. Then, insert p into the B-structure.

2. Using the box tree, find all boxes that contain p. For each reported element $box_0(q)$, follow the pointer to its occurrence in the box list. If the bit of $box_0(q)$ has value *true*, then check if p is contained in the interior of $box(q)$. If so, do the following:

2.a. Search in the dictionary for q. Insert p into $N(q)$, insert $d(q,p)$ into the heap and store with p a pointer to $d(q,p)$.

2.b. If $N(q)$ has size less than $(2D+2)^D$, then the insertion algorithm is completed. Otherwise, let $r_1, \ldots, r_l$ be all points of $N(q)$ that are at maximal L_∞-distance from q. For each $1 \leq i \leq l$, delete r_i from $N(q)$ and delete $d(q, r_i)$ from the heap. Finally, replace $box(q)$—which is stored with $box_0(q)$ in the box list—by the box centered at q having r_1 on its boundary, being the new $box(q)$.

Note that since $A \subseteq \{p \in S : p \in S_0 \wedge box(p) \subseteq box_0(p)\}$, all boxes $box(q)$, $q \in A$, that contain p are found in Step 2.

The deletion algorithm: Let $p \in S$ be the point to be deleted.

1. Search for p in the dictionary. If $p \in B$, then delete p from this dictionary and from the B-structure. Otherwise, if $p \in A$, follow the pointer to $box_0(p)$ in the box list and set its bit to *false*. Moreover, for each point q in $N(p)$, delete the distance $d(p,q)$ from the heap. Then, delete p from the dictionary.

2. Using the box tree, find all boxes that contain p. For each reported element $box_0(q)$, follow the pointer to its occurrence in the box list. If the bit of $box_0(q)$ has value *true*, then check if p is contained in $box(q)$. If so, do the following: Set the bit of $box_0(q)$ to *false*. Search in the dictionary for q. For each point r in $N(q)$, delete $d(q,r)$ from the heap. Then, delete the pointer from q to $box_0(q)$, delete the list $N(q)$, and store with q a bit saying that it belongs to B. Finally, insert q into the B-structure.

Rebuild: Recall that n_0 is the size of S at the moment we initialize the structure. After $f(n_0)$ updates have been performed, we discard the entire structure and initialize a new data structure for the current S.

Theorem 7 *Let DS be any data structure for the dynamic closest pair problem. Let $S(n)$ and $U(n)$ denote the size and update time of DS, respectively. Let $1 \leq f(n) \leq n/2$ be a non-decreasing integer function. We can transform DS into another data structure for the dynamic closest pair problem having size $O(n + S(f(n)))$, and an amortized update* *time of* $O((\log n)^{D-1} + U(f(n)) + (n \log n)/f(n))$.

If we apply this theorem twice (for the case $D \geq 3$), respectively k times (for the case $D = 2$), with appropriate choices for the function $f(n)$, then we get the results given in the last three lines of Table 1.

References

[1] S. Arya and D.M. Mount. *Approximate nearest neighbor queries in fixed dimensions.* Proc. SODA 1993, 271-280.

[2] S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, A. Wu. *An optimal algorithm for approximate nearest neighbor searching.* Proc. SODA 1994, pp. 573-582.

[3] K.L. Clarkson. *A randomized algorithm for closest-point queries.* SIAM J. Comput. **17** (1988), 830-847.

[4] H. Edelsbrunner, G. Haring and D. Hilbert. *Rectangular point location in d dimensions with applications.* The Computer Journal **29** (1986), 76-82.

[5] M. Golin, R. Raman, C. Schwarz and M. Smid. *Randomized data structures for the dynamic closest-pair problem.* Proc. SODA 1993, 301-310.

[6] K. Mehlhorn and S. Näher. *Dynamic fractional cascading.* Algorithmica **5** (1990), 215-241.

[7] F.P. Preparata and M.I. Shamos. *Computational Geometry, an Introduction.* Springer-Verlag, New York, 1985.

[8] J.S. Salowe. *Shallow interdistance selection and interdistance enumeration.* International Journal of Computational Geometry & Applications **2** (1992), 49-59.

[9] C. Schwarz. *Data structures and algorithms for the dynamic closest pair problem.* Ph.D. Thesis. Universität des Saarlandes, Saarbrücken, 1993.

[10] C. Schwarz, M. Smid and J. Snoeyink. *An optimal algorithm for the on-line closest pair problem.* Proc. ACM Symp. on Computational Geometry, 1992, 330-336.

[11] M. Smid. *Rectangular point location and the dynamic closest pair problem.* Proc. 2nd Annual International Symp. on Algorithms, Lecture Notes in Computer Science, Vol. 557, Springer-Verlag, Berlin, 1991, 364-374.

[12] M. Smid. *Maintaining the minimal distance of a point set in polylogarithmic time.* Discrete Comput. Geom. **7** (1992), 415-431.

[13] K.J. Supowit. *New techniques for some dynamic closest-point and farthest-point problems.* Proc. SODA 1990, 84-90.

[14] P.M. Vaidya. *An* $O(n \log n)$ *algorithm for the all-nearest-neighbors problem.* Discrete Comput. Geom. **4** (1989), 101-115.

[15] A.C. Yao. *On constructing minimum spanning trees in k-dimensional spaces and related problems.* SIAM J. Comput. **11** (1982), 721-736.

Competitive Searching in a Generalized Street

Amitava Datta* Christian Icking**

Abstract

We consider the problem of a robot which has to find a path in an unknown simple polygon from one point s to another point t, based only on what it has seen so far. A *Street* is a polygon for which the two boundary chains from s to t are mutually weakly visible, and the set of streets was the only class of polygons for which a competitive search algorithm was known.

We define a new, strictly larger class of polygons, called *generalized streets* or $\mathcal{G}$-streets which are characterized by the property that every point on the boundary of a $\mathcal{G}$-street is visible from a point on a horizontal line segment connecting the two boundary chains from s to t. We present an on-line strategy for a robot placed at s to find t in an unknown rectilinear $\mathcal{G}$-street; the length of the path created is at most 9 times the length of the shortest path in the L_1 metric. This is optimal since we show that no strategy can achieve a smaller competitive factor for all rectilinear $\mathcal{G}$-streets. Compared to the L_2-shortest path, the strategy is 9.06-competitive which leaves only a very small gap to the lower bound of 9.

1 Introduction

Path planning is one of the fundamental problems in robotics. In contrast to path planning with complete information, i. e. if the whole scene is known in advance, it is interesting for many real life situations to consider the problem of finding a path on-line. Here, the geometry of the scene is unknown and the robot has to make decisions depending on the information it gathers through (e. g. visual or tactile) sensors.

Lumelsky and Stepanov [8] study this problem when only tactile sensors are used. Papadimitriou and Yannakakis [9] consider several variants of the problem as a two-person game and gave bounds on the length of the generated path in terms of the length of the shortest path. Blum *et al* [2] present deterministic and randomized on-line algorithms for several versions of this problem when the obstacles are rectangles or convex polygons.

Following the concept of *competitive algorithms* introduced by Sleator and Tarjan [10] for general problems in computer science, a strategy for searching on-line is called *k-competitive* for a constant k, if it can be guaranteed that the generated path is no longer than k times the shortest path. The constant k is then called the *competitive factor.*

Klein [6] describes the problem of designing an on-line strategy for a robot when the robot is constrained to move inside a simple polygon from a vertex s to a vertex t. He gives an on-line strategy for a class of polygons called *streets.* A street is a polygon such that the clockwise polygonal chain L from s to t and the anticlockwise polygonal chain R from s to t are mutually weakly visible. An alternative criterion for recognizing a street was also given in [5]. Das *et al* [3] give an algorithm to compute, for a given polygon, all pairs of points s and t such that the polygon is a street with respect to these points.

Klein proves a lower bound of $\sqrt{2}$ (> 1.41) on the competitive factor for searching in a street, and his strategy achieves a competitive factor of $1+\frac{3}{2}\pi$ (< 5.72). Recently, Kleinberg [7] has improved this ratio to $2\sqrt{2}$ (< 2.83). He also mentions that his strategy is $\sqrt{2}$-competitive for rectilinear streets which is optimal. Interestingly, this means in fact that one can find on-line an L_1-shortest path for rectilinear streets.

It has been posed as a challenging open problem whether more general classes of polygons admit competitive searching. In this paper, we introduce one such class.

The rest of the paper is organized as follows. In Section 2, we introduce some definitions and properties related to $\mathcal{G}$-streets. We give the strategy for rectilinear $\mathcal{G}$-streets and analyze its complexity in Section 3. Some open questions are mentioned in Section 4.

*This work was done when the author was visiting FernUniversität Hagen, and the author will like to thank Rolf Klein for his support.

**Praktische Informatik VI, FernUniversität Hagen,
Elberfelder Straße 95, D-58084 Hagen, Germany.
email: christian.icking@fernuni-hagen.de

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

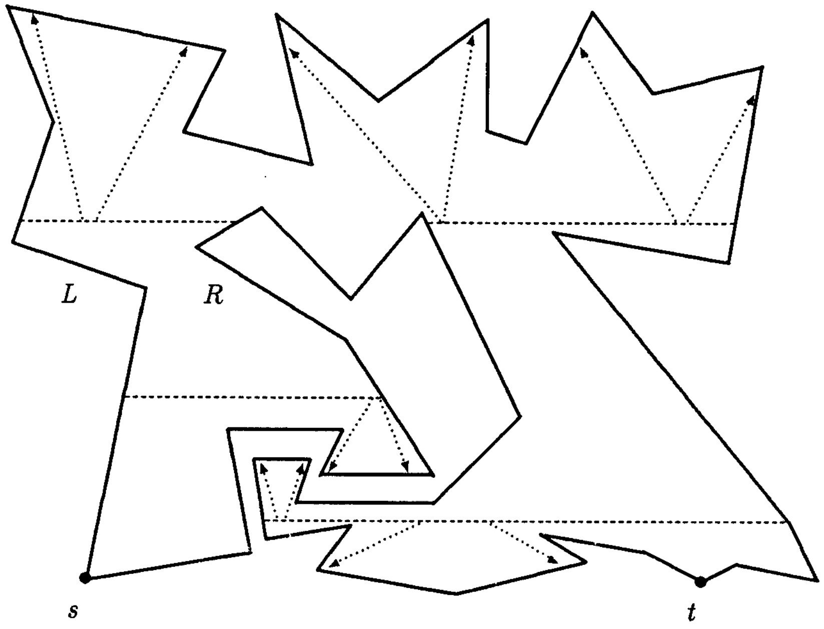


Figure 1: A $\mathcal{G}$-street.

2 Definitions and properties

We consider a simple polygon P in the plane with a start vertex, s, and a target vertex, t. We use the Cartesian coordinate system. The notion of *horizontal* (resp. *vertical*) indicates a direction parallel to the x-axis (resp. y-axis).

We assume for simplicity that no three vertices are aligned. Our results can be easily adapted to handle the general case when three vertices are aligned or s and t are not vertices. The clockwise (resp. anticlockwise) chain from s to t is called the *left chain* or L (resp. *right chain* or R). These two chains induce a natural ordering of vertices of the polygon from s to t.

Definition 1 [6] A polygon is called a *street* if the two chains L and R are mutually weakly visible.

Definition 2 A horizontal line segment inside the polygon P with both endpoints on the boundary of P is called a *chord.* The extension of a horizontal edge e across the interior of P such that the two end points hit the polygon boundary is called the *chord of* e, analogously we define the *chord of a vertex.* If a chord touches both the L and R chains it is called an *LR-chord.*

Definition 3 A simple polygon in the plane is called a *generalized street* or *$\mathcal{G}$-street* if for every boundary point $p \in L \cup R$, there exists an LR-chord c such that p is visible from a point on c.

Figure 1 shows an example of a $\mathcal{G}$-street. Some LR-chords, drawn as dashed lines, indicate visibilities along the dotted rays.

Lemma 4 *The class of $\mathcal{G}$-streets contains all streets and this containment is proper.*

Proof. Consider a street and a point p in its left chain L, see Figure 2. We want to show that p is visible from a point on an LR-chord. Within the street, p must be visible from some point $q \in R$. We shoot a horizontal ray from p into the interior of the street. If it hits R then p lies on an LR-chord and we are done.

Otherwise we move a point r from p to q on the line segment $\overline{pq}$ from which we continue to shoot horizontal rays in both directions, see Figure 2. We stop at the first position where a ray touches the right chain R, which must eventually happen since $q \in R$. This ray determines an LR-chord from which p is visible.

This proves that every street is a $\mathcal{G}$-street. In Figure 1, we have seen an example of a $\mathcal{G}$-street which is not a street. □

The following lemma states a simple property of LR-chords.

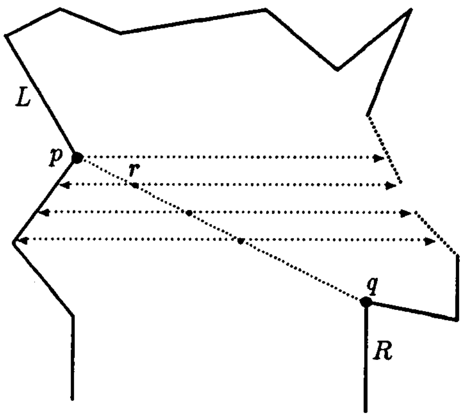


Figure 2: A street is always a $\mathcal{G}$-street (Lemma 4).

Lemma 5 *Every path in P from s to t intersects all LR-chords of the polygon.*

Proof. Consider an LR-chord c and suppose c touches the chains L and R at the points p and q respectively. The line segment $\overline{pq}$ divides the polygon into two parts. The point t cannot be in the part which also contains s, otherwise p and q would be on the same chain L or R. This implies that s and t are in different parts and any path from s to t must intersect $\overline{pq}$. □

3 A strategy for rectilinear $\mathcal{G}$-streets

From now on, we concentrate on rectilinear $\mathcal{G}$-streets; all edges are either horizontal or vertical, see Figure 3 for an example. It is clear that the class of rectilinear $\mathcal{G}$-streets includes all rectilinear streets and is strictly larger.

Definition 6

- A horizontal edge of the polygon is called a *cut edge* if both of its vertices are reflex. The chord of a cut edge is called a *cut chord.*
- A horizontal edge is called a *step edge* if one of its vertices is convex and the other is reflex. The chord of a step edge is called a *step chord.*
- A cut chord divides the polygon into three parts, a step chord divides it into two parts. We call these parts the *regions* of the chord.

We refer to a cut chord or step chord as a *critical chord,* see Figure 4 for examples. It is easy to see that iff the two points s and t are in two different regions of a chord then it is an LR-chord.

Definition 7 A reflex vertex l is called a *landmark with respect to an LR-chord* c if

- l is visible from some point of c, and
- both, the vertical edge incident on l, and c, are part of the same region of the chord of l.

See Figure 4 for an example of a landmark. A landmark is always an endpoint of a cut chord or a step chord. For the robot, to see a landmark will be an evidence that the target is hidden behind the chord of that landmark.

Lemma 8 *Let l be a landmark with respect to an LR-chord and let c be the chord of the horizontal (cut or step) edge of l. Then the following is true.*

1. *The points s and t are in different regions of c, i. e. c is an LR-chord.*
2. *All landmarks (if any) with respect to c are in the regions of s and t.*
3. *Either the point t or some landmark in the region of t is visible from some point on c.*

Proof. We prove the lemma for the case of c being a cut chord. The proof for a step chord is similar.

Suppose, c is a cut chord which passes through the cut edge e. Suppose, the vertex v is adjacent to e and v is a landmark with respect to an LR-chord c_1. First,

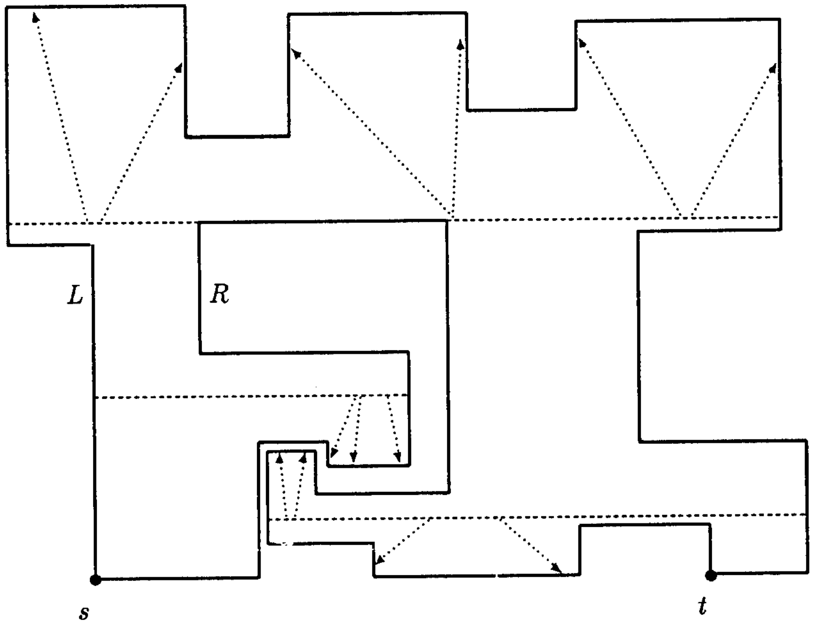


Figure 3: A rectilinear $\mathcal{G}$-street.

assume that s and t are in the same region of c. If s and t are in the region which does not contain c_1, then c_1 is not an LR-chord. If they are in the same region which contains c_1, then there is at least one point on the edge e which is not visible from any LR-chord and the polygon is not a $\mathcal{G}$-street, a contradiction. Hence, c is an LR-chord and the two points s and t are in two different regions of the chord c.

For proving the second statement, assume that there is a landmark l_2 on a horizontal edge e_2 in a region R_2 other than the ones which contain s and t. Note that no horizontal line segment (except the chord c) whose end points touch the polygon in the region R_2 can be an LR-chord. This is because the complete polygonal chain which constitutes this region belongs either to the L or to the R chain. So, there is at least one point on e_2 which is not visible from any LR-chord. This means the polygon is not a $\mathcal{G}$-street, a contradiction.

For the third statement, consider the two regions which do not contain the point s. We have already proved that t must be in one of these two regions. So, t can be hidden from all the points of the chord c only through the presence of a landmark. □

Lemma 9 *If c is the chord of the starting point s then the following holds.*

- *The point t or some landmark in the region of t is visible from some point on c.*
- *The only visible landmarks are in the region of T.*

Proof. Similar to that of Lemma 8. □

Now, we turn to a description of our algorithm. We assume that at every position in its path, the robot can get the visibility map of the polygon through its sensors. The idea for our strategy is inspired by the strategy, called *doubling*, for searching a point on a line by Baeza-Yates *et al* [1]. There, the robot goes back and forth on the line, at each step doubling the distance to the start point, until the target is reached.

Lemma 10 [1] *The* doubling *strategy for searching a point on a line has a competitive factor of* 9, *and this is optimal.*

For the problem of finding a path in a $\mathcal{G}$-street, we apply a slight variant of the doubling strategy. Standing on a critical LR-chord, our robot performs doubling on this chord until the target becomes visible or a landmark is in sight. If at any time during this doubling a vertical wall is hit, the robot walks straight in the other direction. Also, it may happen that after reaching a cut

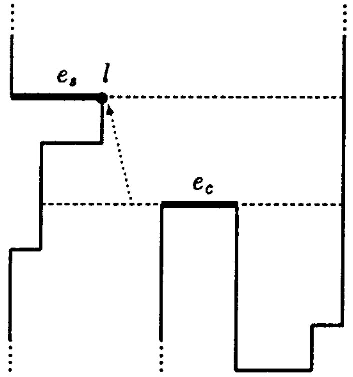


Figure 4: A *step edge* e_s, a *cut edge* e_c, and the respective chords. Vertex l is a *landmark* with respect to the chord of e_c.

or step chord, the robot immediately sees the next landmark and it does not execute doubling on this chord. We do not differentiate between all these variants of the doubling algorithm in our strategy and call all of them *doubling.*

The doubling strategy in [1] is formulated to search for the target only on an integer grid. It is not competitive if the distance to the target may be any real number, because the first step may be to the wrong direction. Nevertheless, we can use doubling in our environment of real coordinates if we chose as length of the first step for each doubling the distance of the actual position of the robot to the nearest projection of all visible vertices on the actual chord.

Our algorithm, called *doubleX*, is presented (in pseudo-code) in Table 1. The robot walks on a rectilinear path and tries to keep close to an L_1-shortest path. Initially, when the robot is at the point s, it either sees a landmark or the goal by executing the doubling algorithm on the chord of s. This is possible due to Lemma 9. So, initially the robot can choose a region depending on this landmark. At every successive step, the robot moves from one LR-chord defined by a landmark to another LR-chord defined by the next landmark. These landmarks can be correctly chosen due to Lemma 8. If there is more than one visible landmark then the robot is free to chose one of them. In this process, ultimately the goal is reached.

We need the following lemma before proving the feasibility of step 10 of the algorithm.

Lemma 11 *If a landmark l is visible from the present position, the robot can reach the L_1-nearest point on the chord of l by an xy-monotone, i.e. L_1-shortest, path.*

Proof. We assume w.l.o.g. that the landmark l is on the right chain R and to the right and above the present position p of the robot, see Figure 5. The other cases are similar.

The robot goes vertically upwards from its present position until it reaches the chord of the landmark l (then we are done), or l becomes invisible. The visibility ray to l can only be obstructed by a reflex vertex of L. At this point, the robot starts going to the right until it is vertically below a point of the chord of l, or l becomes invisible again (whatever comes first).

This process is repeated until the chord of l is reached, see the thick line in Figure 5. Clearly, the resulting path is xy-monotone.

The reached point on the chord of l either is vertically above p, or has an x-coordinate equal to the maximum x-coordinate of the subchain of L between the chord of p and the chord of l. This shows that we reach the chord of l at the L_1-nearest point from p. □

It is easy to see that if t is visible from the present position of the robot, it can reach t in a similar way, so that step 15 of the algorithm is feasible.

Lemma 12 *Starting from the point s, the robot correctly reaches the point t by executing procedure doubleX.*

Proof. Initially, there are two possibilities at the point s. If the robot sees the point t, it can go to t by a method similar to that in Lemma 11. Otherwise, it finds a landmark according to Lemma 9. From Lemma 5, the path should cross the chord which passes through this landmark. So, initially, the robot chooses a correct

```
Procedure doubleX
s : the starting point.
t : the target point.
m : a chord.
begin
   Let m be the chord of s.
   Execute doubling on m until a landmark l or the target t is visible.
   while (t is not visible) do
     Let m be the chord of l.
     Go to the $L_1$-nearest point of m on an xy-monotone path.
     Determine the region R of m where we come from.
     Execute doubling on m
         until a landmark l or the target t is visible in a region other than R.
   end while
   Go to t on an xy-monotone path.
end
```

Table 1: The strategy for rectilinear $\mathcal{G}$-streets.

vertical direction. In every subsequent step, the robot reaches a critical chord. While executing the doubling algorithm at each such chord, it is always ensured from Lemma 8 that either the next landmark or the point t is found. As soon as the point t is visible, the robot can go to t by a xy-monotone path as mentioned before. □

We now estimate the length of the path generated by the robot.

Definition 13 The *reduced path* is a rectilinear path from s to t which only includes from the robot's path

- the xy-monotone motion from one critical chord to the next, and
- the horizontal step from the point where a critical chord is first reached to the point where it is left.

In other words, for the reduced path, we do not consider the extra movements generated due to the execution of the doubling algorithm.

Lemma 14 *The reduced path is an L_1-shortest path from s to t.*

Proof. Along its way from s to t, the robot stops at a sequence of critical chords $c_0, \ldots, c_i, \ldots, c_k$, where c_0 is the chord of s, and c_k is the last chord the robot crosses before reaching t.

By induction, assume that the path is a shortest path upto the chord c_i. Clearly, this is true for $i = 0$. Now, we have to prove that the path up to c_{i+1} is shortest. If $i = 0$ or if the reduced path from c_{i-1} to c_{i+1} is xy-monotone, this follows from the induction hypothesis.

Otherwise, we have two possible cases. If the reduced path changes its vertical direction at c_i ("up" from c_{i-1} to c_i and "down" from c_i to c_{i+1} or vice versa) then c_i is a cut chord, and every shortest path from s to t goes along the edge of c_i.

If, on the other hand, the horizontal direction changes, then the path from c_{i-1} to c_i must touch a vertical edge at an extreme position such that every shortest path also touches that edge, since otherwise the robot's path, from c_{i-1}, would not reach the L_1-nearest point on c_i.

From this we conclude that the reduced path is shortest up to t. □

Theorem 15 *Procedure* doubleX *achieves a competitive factor of* 9 *in the L_1 metric, and this is optimal.*

Proof. From Lemma 14, the reduced path of the robot is a shortest path. Compared to a reduced path, the only extra path segments the robot traverses are the segments generated due to doubling on a step or cut chord. Suppose, for such a chord c_i, the robot reaches and leaves c_i at the points p_1 and p_2 respectively. From Lemma 10, the robot traverses at most 9 times the length of the segment $\overline{p_1 p_2}$ due to doubling. Hence, the claim follows.

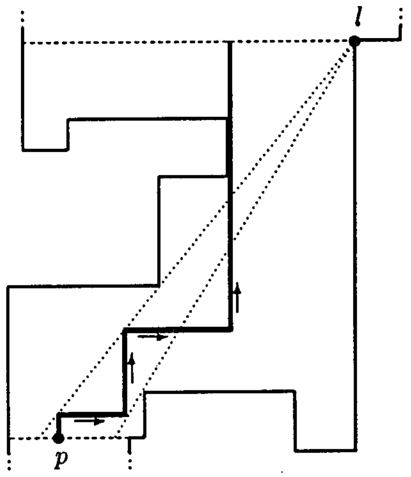


Figure 5: An xy-monotone step from one LR-chord to another.

On the other hand, no better competitive factor can be achieved. Figure 6 shows how we can produce a $\mathcal{G}$-street with many caves at integer positions where t can be in any of the caves, such that searching for t in such a $\mathcal{G}$-street is essentially equivalent to searching a point on a line, for which [1] proves a lower bound of 9. □

The lower bound of 9 also applies if we compare the robot's path to the L_2-shortest path. But what is the competitive factor of our strategy in this case? Any L_1-shortest path is at most $\sqrt{2}$ times longer than the corresponding L_2-shortest path. From this one can conclude that our strategy *doubleX* achieves a competitive factor of $9\sqrt{2}$ (< 12.73) in the L_2 metric. But with a closer look to the details of our strategy, one can prove a much better bound.

Theorem 16 *Procedure* doubleX *achieves a competitive ratio of* 9.06 *in the* L_2 *metric.*

Proof. We consider two reflex vertices p_1 and p_2 of the robots path such that the reduced path from p_1 to p_2 is xy-monotone but the reduced path from p_1 to the critical chord after p_2 (if it exists) is not xy-monotone, as well as the reduced path from the preceding chord of p_1 to p_2.

Let x_1 be the x-distance between p_1 and p_2 and y_1 the y-distance (such that the reduced path from p_1 to p_2 has length $x + y$).

The length of the L_2-shortest path from p_1 to p_2 is at least $\sqrt{x_1^2 + y_1^2}$. From the proof of Theorem 15 we know that the length of the robot's path is at most $9x_1 + y_1$. We may assume that $y_1 \neq 0$ and substitute $w = x_1/y_1$. The competitive factor must be less than the maximum value of

$$\frac{9x_1 + y_1}{\sqrt{x_1^2 + y_1^2}} = \frac{9w + 1}{\sqrt{w^2 + 1}}$$

for all possible values of w. A simple analysis shows that the maximum is reached at $w = 9$ at which

$$\sqrt{82} < 9.06$$

is the maximum value. The claim of the theorem follows by extending the analysis to every pair of such chords and from the fact that the L_2-shortest path must also visit the points p_1 and p_2. □

4 Conclusion

For the rectilinear case, we have extended the class of polygons for which competitive strategies are known for a robot, starting at a vertex s, to find a path to a vertex t with the help of the visibility map. Our strategy *doubleX* achieves an optimal competitive factor of 9 in the L_1 metric, and a factor of 9.06 in the L_2-metric, which is very close to optimal.

It is an interesting open question whether there is, in the rectilinear case, a still larger class of polygons which can be searched competitively, and in the general case, if the class of $\mathcal{G}$-streets or another class of polygons which strictly includes streets permits a competitive search.

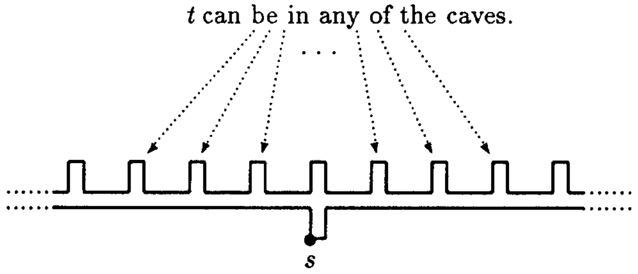


Figure 6: Searching in a $\mathcal{G}$-street is at least as difficult as searching a point on a line.

References

[1] R. Baeza-Yates, J. Culberson, and G. Rawlins. *Searching in the plane.* Information and Computation **106**, 1993, pp. 234–252.

[2] A. Blum, P. Raghavan and B. Schieber. *Navigating in unfamiliar geometric terrain.* In Proceedings of the 23rd ACM Symposium on Theory of Computing, 1991, pp. 494–504.

[3] G. Das, P. J. Heffernan, and G. Narasimhan. *LR-Visibility in Polygons.* In Proceedings of the 5th Canadian Conference on Computational Geometry, 1993, pp. 303–307.

[4] X. Deng, T. Kameda, and C. Papadimitriou. *How to learn an unknown environment I: The rectilinear case.* Technical Report CS-93-04, Department of Computer Science, York University, 1993.

[5] Ch. Icking and R. Klein. *The Two Guards Problem.* International Journal of Computational Geometry and Applications **2**, 1992, pp. 257–285.

[6] R. Klein. *Walking an unknown street with bounded detour.* Computational Geometry: Theory and Applications **1**, 1992, pp. 325–351.

[7] J. Kleinberg. *On-line search in a simple polygon.* In Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms, 1994, pp. 8–15.

[8] V. J. Lumelsky and A. A. Stepanov. *Path-planning strategies of a point mobile autmaton moving amidst unknown obstacles of arbitrary shape.* Algorithmica **2**, 1987, pp. 403–430.

[9] C. Papadimitriou and M. Yannakakis. *Shortest paths without a map.* Theoretical Computer Science **84**, 1991, pp. 127–150.

[10] D. D. Sleator and R. E. Tarjan. *Amortized Efficiency of List Update and Paging Rules.* Communications of the ACM, **28**, 1985, pp. 202–208.

The Tight Lower Bound for the Steiner Ratio in Minkowski Planes

Biao Gao* Ding-Zhu Du* Ronald L. Graham†

Abstract

A minimum Steiner tree for a given set X of points is a network interconnecting the points of X having minimum possible total length. The Steiner ratio for a metric space is the largest lower bound for the ratio of lengths between a minimum Steiner tree and a minimum spanning tree on the same set of points in the metric space. In this note, we show that for any Minkowski plane, the Steiner ratio is at least 2/3. This settles a conjecture of D. Cieslik, and also Du et al..

1 Introduction

Given a compact, convex, centrally symmetric domain D in the Euclidean plane E^2, one can define a norm $\|\cdot\|_D : E^2 \to R$ by setting $\|\bar{x}\|_D = \lambda$ where $\bar{x} = \lambda\bar{u}$ and $\bar{u} \in \partial D$, the boundary of D. We can then define a metric d_D on E^2 by taking

$$d_D(\bar{x},\bar{y}) = \|\bar{x}-\bar{y}\|_D.$$

Thus, $\partial D = \{\bar{x} \,|\, \|\bar{x}\|_D = 1\}$. The resulting metric space $M = M(D) = (E^2, d_D)$ is often called a *Minkowski* or *normed* plane with unit disc D. We will usually suppress the explicit dependence of various quantities on D. For a finite subset $X \subset E^2$, a minimum spanning tree $S = S(X)$ consists of a collection of segments AB with A, $B \in X$, which spans all the points of X, and such that the sum of all the lengths $\|AB\|_D$ is a minimum. We denote this minimum sum by $L_M(X)$. Further, we define

$$L_S(X) = \inf_{Y \supseteq X} L_M(Y)$$

where Y ranges over all finite subsets of E^2 containing X. It is not hard to show that there always exists $X' \supseteq X$ with $|X'| \leq 2|X| - 2$ having $L_S(X) = L_M(X')$. When equality holds we say that the Steiner tree $T(X)$ ($= S(X')$) is a *full* Steiner tree for X. The minimum spanning tree $S(Y)$ will be called a *minimum Steiner tree* $T(X)$ for X. The points of $Y\backslash X$ are usually called *Steiner* points of $T(X)$; the points of X are known as *regular* points of $T(X)$.

Minimum Steiner trees have been the subject of extensive investigations during the past 25 years or so (see[4, 11, 16, 9]). Most of this research has dealt with the Euclidean metric, with much of the remaining work dealt with the L_1 metric, or more generally, the usual L_p metric or norm (see [6, 3]). It has been shown, for example, that the determination of $L_S(X)$ in general is an NP-complete problem, both for the Euclidean as well as the L_1 case (cf. [10], [9]).

In this note, we study the *Steiner ratio* $\rho(D)$ for $M(D)$, defined by

$$\rho(D) := \inf_X \frac{L_S(X)}{L_M(X)}.$$

Thus, $\rho(D)$ is a measure of how much the total length of a minimum spanning tree can be decreased by allowing additional (Steiner) points. It is known that for L_1 metric (so that D is the square with vertices $(\pm 1, 0)$, $(0, \pm 1)$), $\rho(D) = 2/3$ [13] and for the Euclidean (or L_2) metric, $\rho(D) = \sqrt{3}/2$ [7]. More recently, Cieslik [3] and Du, Gao, Graham, Liu and Wan [6] independently conjectured that for any normed plane,

$$2/3 \leq \rho(D) \leq \sqrt{3}/2.$$

*Computer Science Department, University of Minnesota, Minneapolis, MN 55455, and Institute of Applied Mathematics, Chinese Academy of Sciences, Beijing. Support in part by the NSF under grant CCR-9208913.

†AT&T Bell Laboratories, Murray Hill, NJ 07974

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

Cieslik [3] showed that for any normed plane,

$$0.612 < \rho(D) < 0.9036$$

while Du et al. [6] proved that for any normed plane,

$$0.623 < \rho(D) < 0.8686$$

We will prove here that for any normed plane,

$$\rho(D) \geq 2/3.$$

Since the L_1 metric has $\rho = .2/3$ then this inequality is therefore best possible.

For prior results on minimum Steiner trees in normed planes, the reader should consult [2], [8], [1], [17] and [19]. This note is organized in the following way. In Section 2, fundamental properties of minimum Steiner trees are presented. In Section 3, the main result is proved.

2 Preliminaries

A minimum Steiner tree is *full* if every regular point is a leaf (i.e., has degree one). The following lemma states an important property of full minimum Steiner trees, which can be found in [6]

Lemma 1 *Suppose that ∂D is differentiable and strictly convex. Then every full Steiner minimum tree consists of three sets of parallel segments.*

A tree is called a *3-regular* tree if every vertex which is not a leaf has degree three. A consequence of Lemma 1 is that for strictly convex and differentiable norms, every minimum Steiner tree is a 3-regular tree.

Another consequence of Lemma 1 is the following result. A proof can be found in [5].

Lemma 2 *For strictly convex and differentiable norms, every full minimum Steiner tree on more than three points must have at least one of the local structures shown in Figure 1.*

Consider a full minimum Steiner tree T in a plane with a strictly convex and differentiable norm. Two regular points are called *adjacent* if one can be reached from the other by always moving in a *clockwise* direction or always moving in a *counterclockwise* direction. Clearly, each regular point has two other adjacent regular points.

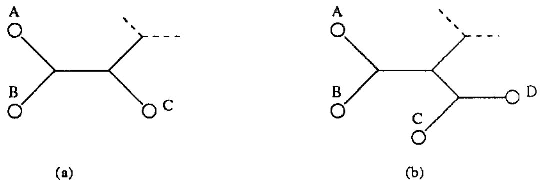


Figure 1: Local structures in full minimum Steiner trees

We can form a polygon G, called the *characteristic* polygon of T, by joining each pair of adjacent regular points with a straight line segment. Any spanning tree lying inside G is called an *inner* spanning tree. A *minimum* inner spanning tree is one having the least possible total length. A point set P is called *critical* if there is a minimum Steiner tree T for P such that the union of the minimum inner spanning trees (with respect to T) for P divides the characteristic polygon $G = G(T)$ into equilateral triangles. The vertices of these equilateral triangles (which we will call lattice points) lie on a triangular lattice in the normed plane.

Since similar sets have the same ratios of minimum Steiner tree and minimum spanning tree lengths, we need only consider critical sets having equilateral triangles with unit edge length. Clearly, for any critical set, a minimum inner spanning tree is in fact a minimum spanning tree; its length is just $n - 1$ where n is the number of its (regular) vertices. Note that any two adjacent regular points have mutual distance 1.

Define

$$\rho_n(D) := \min_{|P|=n} \frac{L_S(P)}{L_M(P)}.$$

If $\rho_{n-1} > \rho_n$, then n is called a *jump* value. In [7], Du and Hwang prove the following.

Lemma 3 *In a plane with a strictly convex and differentiable norm, if n is a jump value then ρ_n is achieved by some critical set.*

Remark: The proof of Du and Hwang for Gilbert-Pollak conjecture used a contradition argument. In their argument, n is assumed to be the smallest natural numder such that a counterexample of n points exists for the Gilbert-Pollak conjecture. For this n, they proved that ρ is achieved by some critical set and then showed that the Gilbert-Pollak conjecture holds for every critical set. Actually, if assume that n is a jump value, then the argument of Du and Hwang still holds. Thus, we have the above lemma.

3 The Main Result

Theorem 1 *For any convex and centrally symmetric D,*

$$\rho(D) \geq \frac{2}{3}.$$

Moreover, if $\rho_k(D) = 2/3$ for some k, then $k = 4$ and ∂D is a parallelogram.

Proof. To begin with, we first assume that the boundary ∂D of unit disc D is strictly convex and differentiable. Thus, we can apply the results of the preceding section.

Assume that the theorem is false. Let n denote the least value so that $\rho_n(D) < 2/3$. Thus, n is a jump value. By Lemma 3 there exists a critical set P of size n such that

$$\frac{L_S(P)}{L_M(P)} < \frac{2}{3}.$$

that is,

$$L_S(P) < \frac{2}{3} L_M(P) = \frac{2}{3}(n-1). \qquad (1)$$

Let T be a minimum Steiner tree on P which witnesses the criticality of P. We first establish some properties of T.

Lemma 4 *T is a full Steiner tree and every edge of T has length less than $2/3$.*

Proof. If T is not a full Steiner tree, then we can decompose it into two edge-disjoint subtrees T_1 and T_2 which are Steiner trees on point sets P_1 and P_2, respectively, where $P_1 \cup P_2 = P$ and each P_i has size less than n. Thus, by the minilality of n,

$$\begin{aligned} L_S(P) &= L_S(P_1) + L_S(P_2) \geq \frac{2}{3}L_M(P_1) + \frac{2}{3}L_M(P_2) \\ &\geq \frac{2}{3}L_M(P), \end{aligned}$$

contradicting (1).

If T has some edge e of length at least $2/3$, then by removing e, we are left with two vertex-disjoint subtrees T_1 and T_2. Clearly, T_1 and T_2 are Steiner trees on disjoint subsets P_1 and P_2, respectively, where $P_1 \cup P_2 = P$. It follows that

$$\begin{aligned} L_S(P) &\geq L_S(P_1) + L_S(P_2) + \ell(e) \\ &\geq \frac{2}{3}(L_M(P_1) + L_M(P_2) + 1) \\ &\geq \frac{2}{3}L_M(P), \end{aligned}$$

again contradicting (1), where in general we will let $\ell(T)$ denote the total length (under D) of any graph T (such as an edge, path or tree). □

Lemma 5 *Suppose T_1 is a 3-regular subtree of T which has f leaves. Then*

$$\ell(T_1) < \frac{2}{3}(f-1).$$

Proof. Assume that

$$\ell(T_1) \geq \frac{2}{3}(f-1)$$

for some subtree T_1 and suppose that T_1 has r leaves which are regular points. Then the removal of T_1 results in $f-r$ subtrees. Suppose that they interconnect sets of $n_1, n_2, \cdots, n_{f-r}$ regular points, respectively. Then $n_1 + n_2 + \cdots + n_{f-r} = n - r$ and

$$\begin{aligned} & L_S(P) \\ \geq\ & \tfrac{2}{3}(n_1 - 1) + \tfrac{2}{3}(n_2 - 1) + \cdots + \tfrac{2}{3}(n_{f-r} - 1) + \ell(T_1) \\ \geq\ & \tfrac{2}{3}(n-f) + \tfrac{2}{3}(f-1) \\ =\ & \tfrac{2}{3}(n-1), \end{aligned}$$

which contradicts (1). □

Let us call a path $AS_1S_2\cdots S_iB$ joining two adjacent regular points A and B in T *monotone* if it is either a clockwise or counterclockwise path from A to B. We will say that S_1 can be *legally moved* to A if $i \geq 3$ and the subpath $S_1S_2S_3$ can be removed from T (disconnecting it into three subtrees) and replaced by a parallel translate $S_1'S_2'S_3'$ with S_1' located at point A so that $S_1'S_2'S_3'$ intersects $S_3\cdots S_iB$. Thus, the two subtrees containing A and B, respectively, are reconnected by $S_1'S_2'S_3'$.

Lemma 6 *Let $AS_1S_2\cdots S_iB$ be a monotone path in T connecting two regular points A and B. Suppose that S_1 cannot be legally moved to A. Draw a line through B, parallel to AS_1, and intersecting the subpath $S_1S_2S_3$ at B'. Then*

$$\ell(AS_1S_2S_3) + \ell(S_2S_3) - \ell(BB'S_2) \geq 1. \qquad (2)$$

Proof. Since S_1 cannot be legally moved to A, we have $\ell(BB') < \ell(AS_1)$. If B' is on the segment S_1S_2 (see Figure 2(a)), then

$$(\ell(AS_1) - \ell(BB')) + \ell(S_1B') \geq \ell(AB) = 1,$$

that is,

$$\ell(AS_1S_2) - \ell(BB'S_2) \geq 1.$$

Thus, (2) holds.

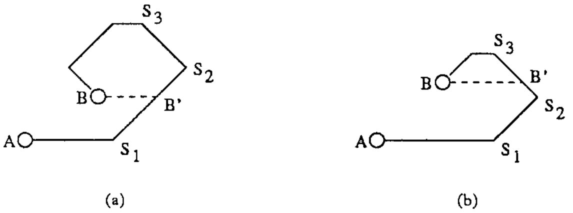


Figure 2:

On the other hand, if B' is on the segment S_2S_3 (see Figure 2(b)), then

$$(\ell(AS_1) - \ell(BB')) + \ell(S_1S_2B') \geq \ell(AB) = 1,$$

that is,

$$\ell(AS_1S_2B') + \ell(S_2B') - \ell(BB'S_2) \geq 1.$$

Thus, (2) also holds in this case, and the lemma is proved. □

It is easy to see that (2) still holds if $\ell(AS_1) = \ell(BB')$.

Lemma 7 *Suppose S_1 is a Steiner point in T adjacent to two regular points A and B. Then S_1 can be legally moved to exactly one of A or B.*

Proof. Let S_2 be the Steiner point adjacent to S_1 and let S_3 and S_4 be the two vertices adjacent to S_2.

Suppose that S_1 can be legally moved to both A and B. Then from these two movements, we can obtain a tree of total length at most $\ell(T) + \ell(S_1S_2)$, which can be decomposed at A and B (see Figure 3). Thus,

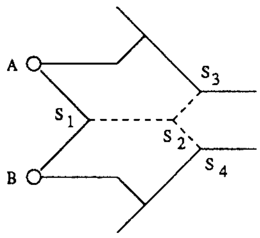


Figure 3:

$$\ell(T) + \ell(S_1S_2) \geq \frac{2}{3}(n-2) + \ell(AS_1B).$$

By Lemma 5,

$$\ell(AS_1B) + \ell(S_1S_2) < \frac{4}{3}.$$

Therefore,

$$\begin{aligned} \ell(T) &\geq \frac{2}{3}(n-2) + \ell(AS_1B) - \ell(S_1S_2) \\ &> \frac{2}{3}(n-4) + 2\ell(AS_1B) \\ &\geq \frac{2}{3}(n-1), \end{aligned}$$

contradicting (1).

Suppose now that S_1 cannot be legally moved to either A or B. Let C be the regular point adjacent to A, other than B, and D the regular point adjacent to B, other than A. By Lemma 6,

$$\ell(AS_1S_2S_3) + \ell(S_2S_3) - \ell(CC'S_2) \geq 1, \quad (3)$$
$$\ell(BS_1S_2S_4) + \ell(S_2S_4) - \ell(DD'S_2) \geq 1, \quad (4)$$

where C' and D' are two points defined in the lemma. Let T' be the 3-regular subtree interconnecting A, B, S_3 and S_4. Adding (3) and (4), we obtain

$$2\ell(T') - \ell(AS_1B) - \ell(CC'S_2D'D) \geq 2,$$

that is,

$$\ell(T') \geq 1 + \frac{1}{2}(\ell(AS_1B) + \ell(CC'S_2D'D)) \geq 2,$$

contradicting Lemma 5. □

We now complete the proof of the first part in the theorem. By Lemma 2, there are two possible local structures we need to consider. We first consider the local structure shown in Figure 1(b). Then there exists a Steiner point S_3 adjacent to two Steiner points S_1 and S_2 each of which is adjacent to two regular points, say A and B are adjacent to S_1, and C and D are adjacent to S_2. Let S_4 be a vertex adjacent to S_3. (See Figure 4.) We first observe that if $\ell(BS_1) = \ell(CS_2)$ then, whether or not S_1 can be legally moved to A, we obtain a contradiction by using the argument given in the proof of Lemma 7. Thus, without loss of generality, we can assume that $\ell(BS_1) > \ell(CS_2)$, i.e., S_1 cannot be legally moved to B. Then by Lemma 6, S_1 can be legally moved to A (see Figure 4). This movement results in a tree of length at most $\ell(T)+\ell(S_3S_4)$, which can be decomposed at A into the subtree AS_1B and a subtree interconnecting $n-1$ regular points other than B. Thus,

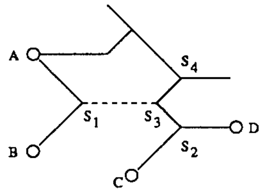


Figure 4:

$$\ell(T)+\ell(S_3S_4) \geq \frac{2}{3}(n-2)+\ell(AS_1B) \geq \frac{2}{3}(n-1)+\frac{1}{3}.$$

Since $\ell(T) < \frac{2}{3}(n-1)$, we have

$$\ell(S_3S_4) > \frac{1}{3}. \qquad (5)$$

Moreover, by Lemma 4, $\ell(BS_1) < \frac{2}{3}$ and $\ell(DS_2) < \frac{2}{3}$. It follows that

$$\ell(CS_2) = \ell(CS_2D) - \ell(DS_2) > \frac{1}{3}.$$

Note that by Lemma 6,

$$\ell(BS_1S_3S_2) - \ell(CS_2) \geq 1.$$

Thus,

$$\ell(S_1S_3S_2) \geq 1 + \ell(CS_2) - \ell(BS_1) > \frac{2}{3}. \qquad (6)$$

Let T' be the 3-regular subtree interconnecting A, B, S_4 and S_2. By (5) and (6),

$$\ell(T') = \ell(AS_1B) + \ell(S_1S_3S_2) + \ell(S_3S_4) > 2,$$

contradicting Lemma 5.

Next, we consider the local structure shown in Figure 1(a), i.e., there exists a Steiner point S_2 adjacent to a Steiner point S_1 and a regular point C such that S_1 is adjacent to two regular points A and B. Let S_3 be the vertex adjacent to S_2, other than C and S_1. We claim that

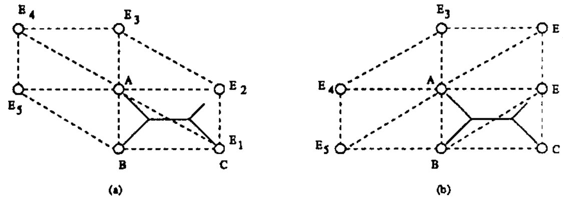


Figure 5:

$$\ell(S_2S_3) < \ell(BS_1). \qquad (7)$$

In fact, if $\ell(S_2S_3) \geq \ell(BS_1)$, then considering the 3-regular subtree T', interconnecting A, B, C and S_3, we would have

$$\ell(T') \geq \ell(BS_1S_2C) + \ell(AS_1B) \geq 2,$$

contradicting Lemma 5. Now, let E be the adjacent regular point of A other than B and let $AS_1 \cdots S_kE$ be the monotone path connecting A and E. From the definition of a critical set, it is easy to see that $\ell(AE) = 1$. Let B, E_1, E_2, $\cdots$, E_5 be all the lattice points with distance exactly one to A (see Figure 5). Then $E \in \{E_1, \cdots, E_5\}$. Since $\ell(AC) < \ell(AB) + \ell(BC) = 2$, C is identical to either E_1 or a lattice point which forms an equilateral triangle with B and E_1 (see Figure 5).

Suppose that E' is a point on the path $S_1S_2S_3$ such that EE' is parallel to AS_1. If E is at E_1, then E' must be on S_2S_3 and $\ell(S_2E') = \ell(BS_1)$. It follows that

$$\ell(S_2S_3) \geq \ell(S_2E') = \ell(BS_1),$$

contradicting (7). The similar argument can be applied to the case that E is at E_2.

If E is at E_3, then it is easy to see $k \geq 4$. Let E'' be a point on $S_2S_3S_4$ such that EE'' is parallel to S_1S_2. Extend BS_1 to F so that EF is parallel to AS_1. Since $\ell(BE) = 2\ell(BA)$, we have $\ell(EF) = 2\ell(AS_1)$ and $\ell(S_1F) = \ell(BS_1)$. Let F' be a point on the path $S_2S_3S_4$ so that FF' is parallel to S_1S_2. If F' is on the segment S_2S_3, then

$$\ell(S_2S_3) \geq \ell(S_1F) = \ell(BS_1),$$

contradicting (7). If F' is on the segment S_3S_4, then

$$\begin{aligned} \ell(S_3S_4) &\geq \ell(EF) = 2\ell(AS_1) \\ &\geq 2(\ell(AB) - \ell(AS_1)) > 2(1 - \frac{2}{3}) = \frac{2}{3}, \end{aligned}$$

contradicting Lemma 4.

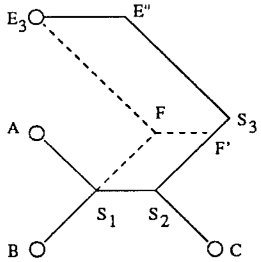


Figure 6:

If E is at E_4 or E_5, then the extension of S_1A must intersect the monotone path $AS_1 \cdots S_kE$. This implies that any line between AS_1 and S_3S_4 and parallel to them must intersect the path $AS_1 \cdots S_kE$. Draw the parallelograms $S_1S_2S_3H$ and AS_1HG and extend HG until it intersects the path $AS_1 \cdots S_kE$, say at F. (AG cannot intersect the path $AS_1 \cdots S_kE$ since otherwise, removing S_2S_3 and adding AG would result in a tree of length at most $\ell(T)$ which does not satisfy the condition in Lemma 1 at the intersection of AG and the path $AS_1 \cdots S_kE$). Then FHS_3S_4 is also a parallelogram. It is easy to see that

$$\ell(GF) \leq \ell(S_3S_4) - \ell(AS_1).$$

Let T' be the 3-regular subtree interconnecting A, B, C and S_3. Consider the tree $(T \backslash T') \cup AGF$ which interconnects $n-2$ regular points. Then,

$$\ell(T) - \ell(T') + \ell(AGF) \geq \frac{2}{3}(n-3).$$

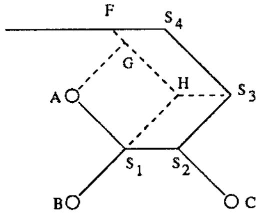


Figure 7:

Moreover,

$$\begin{aligned} &\ell(T') - \ell(AGF) \\ \geq\ & \ell(T') - (\ell(S_2S_3) + \ell(S_3S_4) - \ell(AS_1)) \\ \geq\ & \ell(AS_1B) + \ell(AS_1S_2C) - \ell(S_3S_4) \\ >\ & 2 - \frac{2}{3} = \frac{4}{3}. \end{aligned}$$

Therefore,

$$\ell(T) \geq \frac{2}{3}(n-3) + \ell(T') - \ell(AGF) > \frac{2}{3}(n-1),$$

contradicting (1). This completes the proof of the first part of the theorem for strictly convex and differentiable ∂D.

When ∂D is not strictly convex or not differentiable, we can use a sequence of strictly convex and differentiable ones to approach it from its interior. For each norm in the sequence and for any point set P, we know that

$$L_S(P) \geq \frac{2}{3}L_M(P). \qquad (8)$$

Since $L_S(P)$ and $L_M(P)$ are continuous functions with respect to the norm for fixed P, then letting the sequence go to its limit, we see that (8) holds for the (arbitrary) limiting norm. This completes the proof of the first part of the theorem.

Next, we show the second part of the theorem. Before doing so, we establish three lemmas.

Lemma 8 *Let $A_1A_2 \cdots A_n$ be a path and let OB_{ij} be the unit vector starting from the origin O along directions A_iA_j. If $\ell(A_1A_2 \cdots A_n) = \ell(A_1A_n)$, then the straight-line segment $B_{12}B_{n-1,n}$ is part of ∂D.*

Proof. We prove that all B_{ij}, $i < j$, are on the same straight line. The lemma is a consequence of this fact.

First, consider $n = 3$. Draw the parallelogram $A_1A_2A_3B$. Without loss of generality, assume

$\ell(A_1A_2) \geq \ell(A_1B) = \ell(A_2A_3)$. Let C be a point on A_1A_2 such that $\ell(A_1C) = \ell(A_1B)$. Let E be the intersection point of BC and A_1A_3. Draw line A_2H parallel to BC and intersecting A_1A_3 at H (see Figure 8). Then

$$\ell(A_1E) = \ell(HA_3)$$

and

$$\ell(A_1H) = \ell(A_1E) \cdot \frac{\ell(A_1A_2)}{\ell(A_1C)}.$$

Thus,

$$\begin{aligned} \ell(A_1A_3) &= \ell(A_1H) + \ell(HA_3) \\ &= \ell(A_1E)(1 + \frac{\ell(A_1A_2)}{\ell(A_1C)}) \\ &= \ell(A_1E)(1 + \frac{\ell(A_1A_2)}{\ell(A_2A_3)}). \end{aligned}$$

Since

$$\ell(A_1A_3) = \ell(A_1A_2A_3) = \ell(A_2A_3)(1 + \frac{\ell(A_1A_2)}{\ell(A_2A_3)}),$$

we have

$$\ell(A_1E) = \ell(A_2A_3) = \ell(A_1B) = \ell(A_1C).$$

This means that quadrilateral A_1CEB is similar to quadrilateral $OB_{12}B_{13}B_{23}$. Therefore, B_{12}, B_{23} and B_{13} are collinear. In addition, B_{13} lies between B_{12} and B_{23}.

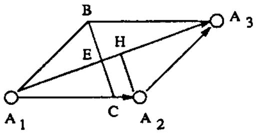


Figure 8:

Next, consider the case $n = 4$. Note that $\ell(A_1A_2A_3A_4) = \ell(A_1A_4)$ implies that $\ell(A_1A_2A_4) = \ell(A_1A_4)$ because

$$\ell(A_1A_2A_3A_4) \geq \ell(A_1A_2) + \ell(A_2A_4) \geq \ell(A_1A_4).$$

From the case $n = 3$, B_{14} is in the segment $[B_{12}, B_{24}]$. Similarly, B_{14} is in the segment $[B_{13}, B_{34}]$, B_{13} is in the segment $[B_{12}, B_{23}]$, and B_{24} is in the segment $[B_{23}, B_{34}]$ (see Figure 9). Note that all B_{ij}'s are on ∂D, a boundary of a convex region. Moreover, for any i, j and k, B_{ij}, B_{jk}, and B_{ik} are either all distinct or all identical. It follows that all B_{ij} for $1 \leq i < j \leq 4$ are collinear.

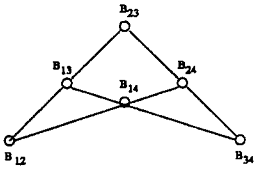


Figure 9:

Now consider $n > 4$. Note that $\ell(A_1A_2 \cdots A_n) = \ell(A_1A_n)$ implies that for $4 \leq j \leq n$, $\ell(A_1A_2A_3A_j) = \ell(A_1A_j)$ and for $3 \leq j < k \leq n$, $\ell(A_1A_2A_jA_k) = \ell(A_1A_k)$. Therefore, for $4 \leq j \leq n$,

$$B_{12}, B_{23}, B_{13}, B_{1j}, B_{2j} \text{ and } B_{3j} \text{ are collinear}$$

and for $3 \leq j < k \leq n$,

$$B_{12}, B_{1j}, B_{2j}, B_{1k}, B_{2k} \text{ and } B_{jk} \text{ are collinear.}$$

It follows that all B_{ij} for $1 \leq i < j \leq n$ are collinear. □

Lemma 9 *$\rho_4(D) = 2/3$ iff ∂D is a parallelogram.*

Proof. Note that for any D,

$$\rho_3(D) \geq 3/4.$$

Suppose $\rho_4(D) = 2/3$. Thus, 4 is a jump value. Consider $F = \{(A, B, C, E) \mid L_M(A, B, C, E) = 1\}$. Since $L_M(A, B, C, E)$ is continuous with respect to A, B, C and E, then F is a compact set in 8-dimensional space. Clearly,

$$\rho_4(D) = \inf_{(A,B,C,E) \in F} L_S(A, B, C, E).$$

Since $L_S(A, B, C, E)$ is also continuous with respect to A, B, C, and E, there exists a point set $\{A, B, C, E\}$ such that

$$2/3 = \rho_4(D) = L_S(A, B, C, E)/L_M(A, B, C, E). \quad (9)$$

Note that the minimum Steiner tree T for this point set must be full because 4 is a jump value. Suppose that A, B, C and E are arranged in the order as shown

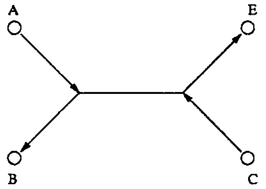


Figure 10:

in Figure 10. Let p_{XY} denote the path from X to Y in T. We claim that the

$$\ell(p_{AB}) = \ell(AB), \quad (10)$$
$$\ell(p_{CB}) = \ell(CB), \quad (11)$$
$$\ell(p_{CE}) = \ell(CE), \quad (12)$$
$$\ell(p_{AE}) = \ell(AE). \quad (13)$$

In fact, if one of them does not hold, then

$$\begin{aligned} & \ell(AB) + \ell(CB) + \ell(CE) + \ell(AE) \\ < \; & \ell(p_{AB}) + \ell(p_{CB}) + \ell(p_{CE}) + \ell(p_{AE}) = 2\ell(T). \end{aligned}$$

So,

$$\begin{aligned} & 4L_M(A,B,C,E) \\ \leq \; & 3(\ell(AB) + \ell(CB) + \ell(CE) + \ell(AE)) \\ < \; & 6\ell(T) = 6L_S(A,B,C,E), \end{aligned}$$

contradicting (9). By Lemma 8 and (10)-(13), ∂D must be a parallelogram. □

Lemma 10 *Let $d(\partial D, \partial D')$ denote the maximum Euclidean distance between the two intersections of a ray from the origin with ∂D and $\partial D'$. Then for any $\delta > 0$ and k, there exists $\epsilon > 0$ such that $d(\partial D, \partial D') < \epsilon$ implies $|\rho_k(D) - \rho_k(D')| < \delta$.*

Proof. Consider any set of k points as a point in $2k$-dimensional space. Let Ω be the point set in $2k$-dimensional space consisting of "points" each of which is a set of k points in the plane with a Euclidean minimum spanning tree of length one. Then Ω is a compact set. In addition, it is easy to see that for any D,

$$\rho_k(D) = \inf_{P \in \Omega} \frac{L_S(P)}{L_M(P)}.$$

Thus, $\rho_k(D)$ is continuous with respect to D. □

Now, suppose to the contrary that ∂D is not a parallelogram and $\rho_k(D) = 2/3$ for some fixed value of k. By Lemma 9, $\rho_4(D) > 2/3$. Thus, there exists k', $4 < k' \leq k$, such that $\rho_{k'-1}(D) > 2/3$ and $\rho_{k'}(D) = 2/3$. Let P be a set of k' points such that $L_S(P)/L_M(P) = 2/3$. Then every minimum Steiner tree for P is full. By Lemma 10, we can choose a sequence of norms D' with strictly convex and differentiable boundary such that $\rho_{k'-1}(D') < \rho_{k'}(D')$. So, the minimum Steiner tree $T(D')$ for P under each norm D' is still full. By Lemma 1, every $T(D')$ is 3-regular and satisfies the condition that all edges of $T(D')$ lie in three directions. Since the number of 3-regular trees with k' leaves is finite, there is a subsequence of $\{T(D')\}$ which converges to a 3-regular tree and satisfies the same condition. It is easy to see that this tree must be a minimum Steiner tree for P under the norm $||\cdot||_D$. By the argument used in the proof of Du and Hwang [7], it follows that P is a critical set. Now, by using the argument in the proof of the first part of the theorem, taking special care of the cases in which equality holds in various inequalities, we eventually obtain a contradiction. This completes the proof of the second part of the claim and the proof is complete. □

4 Discussion

We conjecture that for any norm $||\cdot||_D$, there exists k such that $\rho_k(D) = \rho(D)$. A consequence of this conjecture is that $\rho(D) = 2/3$ iff ∂D is a parallelogram.

The proof techniques used in this paper are different from those in Hwang [13] for proving 2/3 as the Steiner ratio of the rectilinear plane. Graham and Hwang [12] conjectured that m-dimensional rectilinear space has the Steiner ratio $m/(2m-1)$. Although the methods in [13] do not seem to be applicable to proving this conjecture, perhaps the ideas we use here will be of some help. We hope to consider this in the near future.

References

[1] M. Alfaro, M. Conger, K. Hodges, A. Levy,

R. Kochar, L. Kuklinski, Z. Mahmood and K. von Haam, The structure of singularities of Φ-minimizing networks in R^2, *Pac. J. Math.*, 149 (1991) 201-210.

[2] G.D. Chakerian and M.A. Ghandehari, The Fermat problem in Minkowski space, *Geometriae Dedicata* 17 (1985) 227-238.

[3] D. Cieslik, The Steiner ratio of Banach-Minkowski planes, *Contemporary Methods in Graph Theory* (ed. R. Bodendiek), (BI-Wissenschatteverlag, Mannheim, 1990) 231-248.

[4] E.J. Cockayne, On the Steiner problem, *Canad. Math. Bull.* 10 (1967) 431-450.

[5] D.Z. Du, On Steiner ratio conjectures, *Annals of Operations Research*, 33 (1991) 437-449.

[6] D.Z. Du, B. Gao, R.L. Graham, Z.-C. Liu, and P.-J. Wan, Minimum Steiner trees in normed planes, to appear in *Discrete and Computational Geometry.*

[7] D.Z. Du and F.K. Hwang, A proof of Gilbert-Pollak's conjecture on the Steiner ratio, *Algorithmica* 7 (1992) 121-135.

[8] D.Z. Du and F.K. Hwang, Reducing the Steiner problem in a normed space, *SIAM J. Computing,* 21 (1992) 1001-1007.

[9] M.R. Garey, R.L. Graham and D.S. Johnson, The complexity of computing Steiner minimal trees, *SIAM J. Appl. Math.*, 32 (1977) 835-859.

[10] M.R. Garey and D.S. Johnson, The rectilinear Steiner tree problem is NP-complete, *SIAM J. Appl. Math.*, 32 (1977) 826-834.

[11] E.N. Gilbert and H.O. Pollak, Steiner minimal trees, *SIAM J. Appl. Math.* 178 (1968) 1-29.

[12] R.L. Graham and F.K. Hwang, Remarks on Steiner minimum trees, *Bull. Inst. Math. Acad. Sinica*, 4 (1976) 177-182.

[13] F.K. Hwang, On Steiner minimal trees with rectilinear distance, *SIAM J. Appl. Math.*, 30 (1976) 104-114.

[14] G. Lawlor and F. Morgan, Paired calibrations applied to soapfilms, immiscible fluids, and surfaces or networks minimizing other norms, preprint (1991).

[15] Z.C. Liu and D.Z. Du, On Steiner minimal trees with L_p distance, *Algorithmica*, 7 (1992) 179-191.

[16] Z.A. Melzak, *Companion to Concrete Mathematics*, Vol II, Wiley, New York, 1976.

[17] D. Sankoff and P. Rousseau, Locating the vertices of a Steiner tree in an arbitrary metric space, *Math. Program.*, 9 (1975) 240-246.

[18] M. Sarrafzadeh and C.K. Wong, Hierarchical Steiner tree construction in uniform orientations, unpublished manuscripts.

[19] J.M. Smith and M. Gross, Steiner minimal trees and urban service networks, *J. Socio. Econ. Plng.*, 16 (1982) 21-38.

Bounding the Number of Geometric Permutations Induced by k-Transversals

Jacob E. Goodman*
Richard Pollack†
Rephael Wenger‡

Abstract

We prove that a $(k-1)$-separated family of n compact convex sets in $\mathbb{R}^d$ can be met by k-transversals in at most $O(d)^{d^2}\left(\binom{2^{k+1}-2}{k}\binom{n}{k+1}\right)^{k(d-k)}$ or, for fixed k and d, $O\left(n^{k(k+1)(d-k)}\right)$ different order types. This is the first non-trivial bound for $1 < k < d-1$.

Introduction

Let $\mathcal{A}$ be a family of n compact convex sets in $\mathbb{R}^d$. A *line transversal* of the family $\mathcal{A}$ is a line that intersects every member of $\mathcal{A}$. If the sets in $\mathcal{A}$ are pairwise disjoint, then a directed line transversal induces a linear ordering on the sets, determined by the order in which the line intersects them. An undirected line transversal induces a pair of orderings on $\mathcal{A}$ corresponding to the two orientations of the line. Such a pair of orderings induced by a line transversal is called a *geometric permutation* of the set $\mathcal{A}$.

A generalization of the concept of line transversal is a *k-transversal*, a flat, or affine subspace, of dimension k that intersects every member of $\mathcal{A}$. The family $\mathcal{A}$ is *k-separated* [6] if no subset of $\mathcal{A}$ of size greater than $k+1$ has a k-transversal. Equivalently, $\mathcal{A}$ is k-separated if every family of j sets of $\mathcal{A}$ can be strictly separated from every disjoint family of $k+2-j$ sets of $\mathcal{A}$ by a hyperplane.

If $\mathcal{A}$ is $(k-1)$-separated, then each oriented k-transversal f induces an "ordering" on $\mathcal{A}$ as follows. For each convex set $a \in \mathcal{A}$, choose a point $p_a \in f \cap a$. The *order type* of the set of points $\mathcal{P}_{\mathcal{A}} = \{p_a \mid a \in \mathcal{A}\}$ is the family of orientations of all the ordered $(k+1)$-tuples from $\mathcal{P}_{\mathcal{A}}$. Because $\mathcal{A}$ is $(k-1)$-separated, the order type of $\mathcal{P}_{\mathcal{A}}$ is independent of the choice of p_a. This order type represents the order in which f meets the members of $\mathcal{A}$. An unoriented k-transversal induces a pair of order types on $\mathcal{A}$ corresponding to the two orientations of the k-transversal. Such a pair of orderings is also known as a *geometric permutation* of the set $\mathcal{A}$. A more complete discussion of k-transversals and geometric permutations can be found in the survey paper [6].

What is the maximum number of geometric permutations induced by k-transversals for a $(k-1)$-separated family of n compact convex sets in $\mathbb{R}^d$? For line transversals in the plane, the answer, due to Edelsbrunner and Sharir (see [4]), is $2n-2$. In higher dimensions, much less

[*] City College, City University of New York, New York, NY 10031, U.S.A. (jegcc@cunyvm.cuny.edu). Supported in part by NSF grant DMS91-22065 and by NSA grant MDA904-92-H-3069.

[†] Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, U.S.A. (pollack@geometry.nyu.edu). Supported in part by NSF grant CCR91-22103 and by NSA grant MDA904-92-H-3075.

[‡] Ohio State University, Columbus, OH 43210, U.S.A. (wenger@cis.ohio-state.edu). Supported in part by NSA grant MDA904-93-H-3026 and by the NSF Regional Geometry Institute (Smith College, July 1993) grant DMS90-13220.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

is known. Clearly the number of geometric permutations is at most the number of order types of n points in $\mathbb{R}^k$, which is

$$(\frac{n}{k})^{k^2 n(1+O(1/\log(n/k)))} \quad \text{for } n/k \to \infty$$

by results of [5]. Katchalski, Lewis, and Liu give a lower bound of $\Omega(n^{d-1})$ for line transversals in $\mathbb{R}^d$ [7], while Wenger gives an upper bound of $O(n^{2d-2})$ for the same problem [9]. For k-transversals in $\mathbb{R}^d$ only the trivial lower bound of $\Omega(n^{d-k})$, which follows immediately from the results of [7] by "lifting", is known. Cappell *et al.* give a bound of $O(n^{d-1})$ for the number of geometric permutations induced by hyperplane transversals of families of compact convex sets [3]. This paper gives the first polynomial (in n) bound, $O(n^{k(k+1)(d-k)})$, for arbitrary k and d (Theorem 2).

The main new idea that we use to obtain this polynomial bound is the following result, which may be of independent interest.

The orientation of a labeled set S, of $d+1$ points in $\mathbb{R}^d$ is determined by any set of $2^{d+1}-2$ vectors v_P such that for each non-empty proper subset $P \subset S$, v_P is the unit vector normal to some hyperplane separating P from its complement $S \setminus P$ and pointing *toward* P. This fact is not as transparent as it appears and constitutes Theorem 1.

We would like to express our gratitude to Marie-Françoise Roy for bringing to our attention the embedding of the real Grassmannian $\mathbf{G}_{k,d}$ in $\mathbb{R}^{d^2}$ which we use in the proof of Theorem 2 in place of the more familiar embedding in $\mathbb{R}^{\binom{d}{k}}$ using Plücker coordinates. This embedding has the effect of simplifying the presentation of the argument and of making the bound singly rather than doubly exponential in k.

Determining Orientation from Separation

Let $P = \{x_1, \ldots, x_{d+1}\}$ be a labeled set of $d+1$ points in general position in $\mathbb{R}^d$. The *orientation* of P is the sign of the $(d+1) \times (d+1)$ determinant whose ith row consists of 1 followed by the coordinates of x_i. For every partition of P into an ordered pair (P', P'') of non-empty subsets, choose some oriented hyperplane separating P' from P'' with normal vector $v_{P'}$ pointing toward P'. Let H be the set of these $2^{d+1}-2$ separating hyperplanes and let N be the corresponding set of unit normal vectors. The orientation of an ordered d-tuple of vectors in N is the sign of the determinant of the d vectors taken in order as the rows of a $d \times d$ matrix. The order type of N is the correspondence between these d-tuples and their orientation. There are many choices for H, giving rise to sets N with many different order types. Neverthess, we shall show that the order type of N determines the orientation of P.

First, notice that dropping just two normal vectors from N may make it impossible to determine the orientation of P from the order type on N. Choose any partition of $\{1, \ldots, d\}$ into non-empty subsets I' and I'', and let f be a $(d-1)$-flat in $\mathbb{R}^d$. It is possible to place $d+1$ points $R = x_1, \ldots, x_{d+1}$ in f so that any partition other than the one corresponding to $R' = \{x_i \mid i \in I'\}$ and $R'' = \{x_i \mid i \in I''\}$ can be realized by splitting the point set with a $(d-2)$-flat in f. Construct all such splitting $(d-2)$-flats and extend them to oriented hyperplanes in $\mathbb{R}^d$ to generate a set $\tilde{H}$ consisting of $2^{d+1}-4$ separating hyperplanes. Moving one of the $d+1$ points above or below f gives us two $(d+1)$-tuples of points, P and Q, in general position. The set $\tilde{H}$ separates each of P and Q in the same $2^{d+1}-4$ distinct ways, yet P and Q have opposite order types. The two missing hyperplanes separating points corresponding to R' and R'' provide crucial information about the orientations of P and Q.

Assume now that H is a set of $2^{d+1}-2$ oriented hyperplanes separating a $(d+1)$-tuple P as described above, and let N be the set of unit normal vectors of H. Associate the ith member of P with the ith vertex of the standard d-simplex Δ^d in $\mathbb{R}^d$, whose vertices in canonical order are $(0,0,\ldots,0)$, $(1,0,\ldots,0)$, $(0,1,0,\ldots,0)$,..., $(0,\ldots,0,1)$. Notice that the orientation of the vertices in this order is positive. Using N we construct a map ϕ_N from $\partial\Delta^d$, the boundary of the standard simplex, to the unit sphere $\mathbf{S}^{d-1} \subset \mathbb{R}^d$.

Let K be the boundary complex of Δ^d, and L

the barycentric subdivision of K. Each (closed) face κ in K is associated with a vertex w_κ in L. κ contains a set D' of vertices of K and misses another set D''. Let P' and P'' be the subsets of P associated with D' and D'', respectively. Define $\phi_N(w_\kappa)$ to be the unit normal vector of the oriented hyperplane separating P' and P'' with normal pointing toward P'. Extend this map in each face of the barycentric subdivision by mapping each point $x = \sum a_\kappa w_\kappa$, $\sum a_\kappa = 1$, $a_\kappa \geq 0$, to the vector $\sum a_\kappa \phi_N(w_\kappa)$.

Every $(d-1)$-simplex λ of L contains a unique vertex v' of Δ^d and lies on a facet of Δ^d opposite another vertex $v'' \in \Delta^d$. Let h be some hyperplane perpendicular to the line through the points $p', p'' \in P$ corresponding to v', v'' (resp.) and separating p' from p''. For each vertex w of λ, the vector $\phi_N(w)$ points away from the half-space bounded by h containing p'' and toward the half-space containing p'. Thus all the vectors $\phi_N(w)$ must lie in an open half-space and so $\sum a_\kappa \phi_N(w_\kappa)$ is never 0. Normalize $\sum a_\kappa \phi_N(w_\kappa)$ to map x to a point $\phi_N(x)$ on $\mathbf{S}^{d-1}$.

The resulting mapping, $\phi_N : \partial\Delta^d \rightarrow \mathbf{S}^{d-1}$, has a well-defined *degree*, which is (intuitively) the algebraic number of times ϕ_N wraps $\partial\Delta^d$ around $\mathbf{S}^{d-1}$. For basic facts about this degree, see for example [8].

Lemma 1 *Let P be a $(d+1)$-tuple of points in general position in $\mathbb{R}^d$, let H be some set of separating hyperplanes for P, and let N be the set of normals of H. The map ϕ_N has degree +1 if the orientation of P is positive and degree -1 if the orientation of P is negative.*

Proof: If P has negative orientation then we may reflect it and the hyperplanes in H, say by reversing the sign of the first coordinate. Reflection reverses both the orientation of P and the sign of the degree of the mapping ϕ_N. Thus we may assume that P has positive orientation.

For any $(d+1)$-tuple Q of points in general position in $\mathbb{R}^d$ we define the following canonical separating set. Let q^* be the barycenter of the simplex whose set of vertices is Q. For every partition of Q into sets Q' and Q'' there is a unique hyperplane containing Q' which is parallel to some hyperplane containing Q''. Translate this hyperplane to pass through q^* and orient it with normal pointing toward Q'. Let $H^*(Q)$ and $N^*(Q)$ be this canonical set of hyperplanes and their normal vectors, respectively.

We first show that the degree of ϕ_N equals the degree of $\phi_{N^*(P)}$. The set H of separating hyperplanes can be continuously transformed to the canonical separating set $H^*(P)$. Let H_t denote the intermediate sets of separating hyperplanes where $0 \leq t \leq 1$, $H_0 = H$, and $H_1 = H^*(P)$. Let N_t be the set of normals of H_t.

Since the normals in N_t vary continuously with the separating hyperplanes, for each vertex w_κ of the barycentric subdivision the image $\phi_{N_t}(w_\kappa)$ must vary continuously as a function of t. Hence for any point $x = \sum a_\kappa w_\kappa$, the convex combination $\sum a_\kappa \phi_{N_t}(w_\kappa)$ also varies continously as a function of t, and the same holds for its normalization $\phi_{N_t}(x)$. Hence the degree of ϕ_{N_t} is a constant that does not change with t, and, in particular, the degree of $\phi_N = \phi_{N_0}$ equals the degree of $\phi_{N^*(P)} = \phi_{N_1}$.

Let V_d be the set of vertices of Δ^d in canonical order. It is easy to check that the map $\phi_{N^*(V_d)}$ is a homeomorphism of $\partial\Delta^d$ to the unit sphere and has degree +1. We show that the degree of $\phi_{N^*(P)}$ equals the degree of $\phi_{N^*(V_d)}$.

Since P and V_d have the same positive orientation, we can continuously move the points in P to the corresponding points in V_d, keeping them in general position; this can be shown, for example, by an easy inductive argument. Let P_t denote the intermediate sets of points where $0 \leq t \leq 1$, $P_0 = P$, and $P_1 = V_d$. The sets $H^*(P_t)$ and $N^*(P_t)$ are also continuous functions of t. As argued above, this implies that $\phi_{N^*(P_t)}$ is a continuous function of t. Thus the degree of $\phi_{N^*(P)} = \phi_{N^*(P_0)}$ equals the degree of $\phi_{N^*(V_d)} = \phi_{N^*(P_1)}$, which is +1. Since the degree of ϕ_N equals the degree of $\phi_{N^*(P)}$, the orientation of P matches the degree of ϕ_N. □

Lemma 1 establishes that the orientation of P depends upon the degree of ϕ_N. We now show that the degree of ϕ_N depends only upon the order type of N.

Lemma 2 *The order type of N determines the degree of ϕ_N.*

Proof: As before, let K be the boundary complex of the standard d-simplex, Δ^d, and L the barycentric subdivision of K. Let Θ be the set of all unit vectors that are linear combinations of $d-1$ or fewer vectors of N. Choose any point y' on the unit sphere that is not contained in Θ. Since the degree of ϕ_N is $+1$ or -1, the point $y' = \phi_N(x')$ for some point x' on $\partial\Delta^d$. Point x' lies in the interior of some $(d-1)$-face λ of L. Let $w_0, w_1, \ldots, w_{d-1}$ be the vertices of λ.

For any d vectors $v_1, \ldots, v_d$, let $\det(v_1, \ldots, v_d)$ represent the determinant of the matrix with the vectors v_i (in that order) as rows. Choose an $\epsilon > 0$ such that

$$\epsilon < \frac{1}{d}\frac{|\det(v_1, v_2, \ldots, v_d)|}{|\det(v'_1, v'_2, \ldots, v'_d)|}$$

for all $v_i, v'_i \in N$ where $\det(v_1, \ldots, v_d) \neq 0$ and $\det(v'_1, \ldots, v'_d) \neq 0$. Let $\mu = 1/(1+\epsilon+\cdots+\epsilon^{d-1})$, $x = \mu(w_0+\epsilon w_1+\cdots+\epsilon^{d-1}w_{d-1})$, and $y = \phi_N(x)$.

We claim that the order type of $N \cup \{y\}$ is determined by the order type of N. Let $v_1, \ldots, v_{d-1}$ be a set of $d-1$ vectors in N. The orientation of the d-tuple $(y, v_1, \ldots, v_{d-1})$ is the sign of $\det(y, v_1, \ldots, v_{d-1})$. If the vectors v_i are linearly dependent, then the orientation of the d-tuple $(y, v_1, \ldots, v_{d-1})$ is zero. Assume the v_i are linearly independent. By definition,

$$\phi_N(x) = \nu(\phi_N(w_0) + \epsilon\phi_N(w_1) + \cdots + \epsilon^{d-1}\phi_N(w_{d-1})),$$

where ν is chosen so that $\|\phi_N(x)\| = 1$. Thus

$$\det(y, v_1, \ldots, v_{d-1}) = \nu\sum_{i=0}^{d-1}\epsilon^i \det(\phi_N(w_i), v_1, \ldots, v_{d-1}).$$

By our choice of a suitably small ϵ, the sign of this sum is determined by the sign of the first non-zero term, $\epsilon^i \det(\phi_N(w_i), v_1, \ldots, v_{d-1})$, in this sum. The vectors $\phi_N(w_i), v_1, \ldots, v_{d-1}$ are all in N and so the order type of N determines the sign of $\det(\phi_N(w_i), v_1, \ldots, v_{d-1})$. Thus the orientation of every d-tuple in $N \cup \{y\}$ is completely determined by the orientation of the d-tuples in N.

Notice that $y \notin \Theta$. For otherwise, we would have $\det(y, v_1, \ldots, v_{d-1}) = 0$ for some choice of $v_1, \ldots, v_{d-1}$. But then, by the choice of ϵ, it would follow that $\det(\phi_N(w_i), \ldots, v_{d-1}) = 0$ for each i, which would imply that $\phi_N(w_0), \ldots, \phi_N(w_{d-1})$ are linearly dependent, contrary to the choice of the simplex λ.

Partition the $(d-1)$-simplices of L whose image contains y into those whose orientation is preserved and those whose orientation is reversed by ϕ_N. The degree of ϕ_N is then the number of the former minus the number of the latter (see [8], for example). The simplices whose image contains y are determined by the order type of $N \cup \{y\}$, as is their orientation. Thus the degree of ϕ_N is determined by the order type of N. □

We have therefore proved:

Theorem 1 *The orientation of a $(d+1)$-tuple P of points in general position in $\mathbb{R}^d$ is determined by the order type of the normal vectors to any set of oriented hyperplanes separating each nonempty subset of P from its complement.*

Bounding the Number of Geometric Permutations

Let $\mathcal{A}$ be a $(k-1)$-separated family of n compact convex sets in $\mathbb{R}^d$. For every subset $\mathcal{A}'$ of size $k+1$, choose a family $H_{\mathcal{A}'}$ of $2^{k+1}-2$ oriented separating hyperplanes for $\mathcal{A}'$, and let $N_{\mathcal{A}'}$ be the set of unit normals to the members of $H_{\mathcal{A}'}$. Let $H_{\mathcal{A}}$ and $N_{\mathcal{A}}$ be the sets of all the chosen hyperplanes and normal vectors, respectively.

For each oriented k-transversal f, let $f \cap \mathcal{A}$ and $f \cap H_{\mathcal{A}}$ represent the sets obtained by intersecting f with the sets in $\mathcal{A}$ and with the hyperplanes in $H_{\mathcal{A}}$, respectively. Notice that f meets each separating hyperplane transversally. Let M_f be the $d \times d$ matrix which in the standard basis represents orthogonal projection onto f_0, the oriented k-dimensional subspace of $\mathbb{R}^d$ parallel to f. Notice that M_f is symmetric, idempotent and has trace k. In fact, such matrices are in one-to-one correspondence with the k-dimensional subspaces of $\mathbb{R}^d$, which make up the real Grassmannian $\mathbf{G}_{k,d}$. This Grassmannian is an algebraic variety in $\mathbb{R}^{d^2}$ of dimension $k(d-k)$, defined by polynomials of degree at most 2 (see [2] for more details). Let $f(V)$ be the orthogonal projection of any set V of vectors in $\mathbb{R}^d$ onto f_0. Representing V as the columns of a $d \times |V|$ matrix, we see that $f(V)$ is then represented by the columns of $M_f V$. Notice, too, that $f(N_{\mathcal{A}})$ is a set of normals to $f \cap H_{\mathcal{A}}$.

By Theorem 1, two oriented k-transversals, f and f', generate the same order type on $\mathcal{A}$ if for every $(k+1)$-subset $\mathcal{A}'$ the orientation of every k-tuple in $f(N_{\mathcal{A}'})$ and of the corresponding k-tuple in $f'(N_{\mathcal{A}'})$ is the same. Let V be a k-tuple of $N_{\mathcal{A}'}$. The orientation of $f(V)$ is zero if and only if $M_f(V)$ has rank $< k$, and this will be true if and only if the maximal minors of $M_f(V)$ are all zero. Let g_V be the sum of the squares of these maximal minors. g_V is a polynomial of degree $2d$ in the d^2 indeterminates $u_{11}, \ldots u_{dd}$, the coordinates of a generic point of $\mathbb{R}^{d^2}$. By continuity, two k-transversals f and f' meet $\mathcal{A}'$ in the same order type if the subspaces corresponding to M_f and $M_{f'}$ can be connected in $\mathbf{G}_{k,d}$ without meeting the zero set of g_V for any k-element subset V of $N_{\mathcal{A}'}$.

Each k-subset V of $N_{\mathcal{A}'}$ determines such a polynomial g_V of degree $2d$ in $R[u_{11}, \ldots, u_{dd}]$. There are $\binom{2^{k+1}-2}{k}$ such polynomials for each of the $\binom{n}{k+1}$ subsets $\mathcal{A}'$ of $\mathcal{A}$ of size $k+1$, giving a total of

$$s = \binom{2^{k+1}-2}{k}\binom{n}{k+1}$$

polynomials. In [1] it is shown that the number of connected components of all non-empty sign conditions (and hence in our case of the number of connected components on $\mathbf{G}_{k,d}$ of the complement of the zero set of these s polynomials) of s polynomials in $R[X_1, \ldots, X_m]$, each of degree at most d on a variety of dimension m' defined by polynomials of degree at most d, is at most $s^{m'}O(d)^m$. Applying this bound to our polynomials on the variety $\mathbf{G}_{k,d}$, we have proved:

Theorem 2 *A $(k-1)$-separated family of n compact convex sets in $\mathbb{R}^d$ has at most*

$$O(d)^{d^2}\left(\binom{2^{k+1}-2}{k}\binom{n}{k+1}\right)^{k(d-k)}$$

geometric permutations.

Conclusion

In this paper we have generalized arguments given in [9] for bounding the number of geometric permutations of line transversals in $\mathbb{R}^d$. If we pay attention only to the dependence on n, the $O(n^{k(k+1)(d-k)})$ bound given here matches the one in [9] when $k = 1$. Better bounds are known for the special cases of line transversals in the plane ($\Theta(n)$) and of hyperplane transversals in $\mathbb{R}^d$ ($O(n^{d-1})$) [3, 4, 9]. We suspect that the number of geometric permutations for k-flats is at most $O(n^{k(d-k)})$.

References

[1] Basu, S., Pollack, R., and Roy, M.-F. On the number of cells defined by a family of polynomials on a variety. In *Algorithmic Foundations of Robotics*, K. Y. Goldberg, D. Halperin, J.-C. Latombe, and R. H. Wilson, Eds. A. K. Birkhauser, Boston, to appear.

[2] Bochnak, J., Coste, M., and Roy, M.-F. *Géométrie algébrique réelle.* Springer-Verlag, Heidelberg, 1987.

[3] Cappell, S. E., Goodman, J. E., Pach, J., Pollack, R., Sharir, M., and Wenger, R. The combinatorial complexity of hyperplane transversals. In *Proc. 6th Ann. ACM Sympos. Comput. Geom.* (1990), pp. 83–91.

[4] Edelsbrunner, H., and Sharir, M. The maximum number of ways to stab n convex non-intersecting sets in the plane is $2n-2$. *Discrete Comput. Geom. 5* (1990), 35–42.

[5] Goodman, J. E., and Pollack, R. Upper bounds for configurations and polytopes in $\mathbb{R}^d$. *Discrete Comput. Geom. 1* (1986), 219–227.

[6] Goodman, J. E., Pollack, R., and Wenger, R. Geometric transversal the-

ory. In *New Trends in Discrete and Computational Geometry*, J. Pach, Ed. Springer-Verlag, Heidelberg, 1993, pp. 163–198.

[7] KATCHALSKI, M., LEWIS, T., AND LIU, A. The different ways of stabbing disjoint convex sets. *Discrete Comput. Geom. 7* (1992), 197–206.

[8] LEFSCHETZ, S. *Introduction to Topology.* Princeton University Press, Princeton, 1949.

[9] WENGER, R. Upper bounds on geometric permutations for convex sets. *Discrete Comput. Geom. 5* (1990), 27–33.

Applications of the Crossing Number

János Pach[*]
Dept. of Computer Science, City College-CUNY
and Courant Institute, NYU

Farhad Shahrokhi
Dept. of Computer Science, University of North Texas

Mario Szegedy
AT&T Bell Laboratories

Abstract

We show that any graph of n vertices that can be drawn in the plane with no $k+1$ pairwise crossing edges has at most $c_k n \log^{2k-2} n$ edges. This gives a partial answer to a dual version of a well-known problem of Avital–Hanani, Erdős, Kupitz, Perles, and others. We also construct two point sets $\{p_1, \ldots, p_n\}$, $\{q_1, \ldots, q_n\}$ in the plane such that any piecewise linear one-to-one mapping $f : \mathbf{R}^2 \to \mathbf{R}^2$ with $f(p_i) = q_i$ $(1 \leq i \leq n)$ is composed of at least $\Omega(n^2)$ linear pieces. It follows from a recent result of Souvaine and Wenger that this bound is asymptotically tight. Both proofs are based on a relation between the crossing number and the bisection width of a graph.

1 Introduction

A *geometric graph* is a graph drawn in the plane by (possibly crossing) straight-line segments i.e., it is defined as a pair of $(V(G), E(G))$, where $V(G)$ is a set of points in the plane in general position and $E(G)$ is a set of closed segments whose endpoints belong to $V(G)$.

The following question was raised by Avital and Hanani [AH], Erdős, Kupitz [K] and Perles: What is the maximum number of edges that a geometric graph of n vertices can have without containing $k+1$ pairwise disjoint edges? It was proved in [PT] that for any fixed k the answer is linear in n. (The cases when $k \leq 3$ had been settled earlier by Hopf and Pannwitz [HF], Erdős [E], Alon and Erdős [AE], O'Donnel and Perles [OP], and Goddard, Katchalski and Kleitman [GKK].)

In this paper we shall discuss the dual counterpart of the above problem. We say that two edges of G *cross* each other if they have an interior point in common. Let $e_k(n)$ denote the maximum number of edges that a geometric graph of n vertices can have without containing $k+1$ *pairwise crossing* edges. If G has no crossing edges, then it is a planar graph. Thus, it follows from Euler's polyhedral formula that

$$e_1(n) = 3n - 6 \quad \text{for all } n \geq 3.$$

It was shown in [P] that $e_2(n) < 13n^{3/2}$ and that, for any fixed k,

$$e_k(n) = O(n^{2-1/25(k+1)^2}).$$

However, we suspect that $e_k(n) = O(n)$ holds for every fixed k as n tends to infinity. We know that this is true for *convex* geometric graphs [CP] (a geometric graph is convex if the nodes of the graph are on the convex hall of the drawing). Our next theorem brings us fairly close to this bound for arbitrary geometric graphs. (We remark that our proof works just as well for drawings where the edges are arbitrary Jordan curves.)

Theorem 1.1 *Let G be a geometric graph of n vertices, containing no $k+1$ pairwise crossing edges. Then the number of edges of G satisfies*

$$|E(G)| \leq c_k n \log^{2k-2} n,$$

with a suitable constant c_k depending only on k.

The proof is based on a general result relating the crossing number of a graph to its bisection width (see Theorem 2.1). A nice feature of our approach is that we do not use the assumption that the edges of G are

[*]Supported by NSF grant CCR-91-22103, PSC-CUNY Research Award 663472 and OTKA-4269.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

line segments. Theorem 1.1 remains valid for graphs whose edges are represented by arbitrary Jordan arcs with the property that no arc passes through any vertex other than its endpoints.

The same ideas can be used to settle the following problem. Let T_1 and T_2 be triangles in the plane, and let $\{p_1, \ldots, p_n\}$ and $\{q_1, \ldots, q_n\}$ be two n-element point sets lying in the interior of T_1 and T_2, respectively. A *homeomorphism* f from T_1 onto T_2 is a continuous one-to-one mapping with continuous inverse. f is called *piecewise linear* if there exists a triangulation of T_1 such that f is linear on each of its triangles. The *size* of f is defined as the minimum number of triangles in such a triangulation. Recently, Souvaine and Wenger [SW] have shown that one can always find a piecewise linear homeomorphism $f : T_1 \to T_2$ with $f(p_i) = q_i$ $(1 \leq i \leq n)$ such that the size of f is $O(n^2)$. Our next result shows that this bound cannot be improved.

Theorem 1.2 *There exist a triangle T and two point sets $\{p_1, \ldots, p_n\}$,*
*$\{q_1, \ldots, q_n\} \subseteq$ **int** T such that the size of any piecewise linear homeomorphism $f : T \to T$ which maps p_i to q_i $(1 \leq i \leq n)$ is at least cn^2 (for a suitable constant $c > 0$).*

For some closely related problems consult [S] and [ASS].

2 Crossing number and bisection width

Let G be a graph of n vertices with no loops and no multiple edges. For any partition of the vertex set $V(G)$ into two disjoint parts V_1 and V_2, let $E(V_1, V_2)$ denote the set of edges with one endpoint in V_1 and the other endpoint in V_2. Define the *bisection width* of G as

$$b(G) = \min_{|V_1|, |V_2| \geq n/3} |E(V_1, V_2)|,$$

where the minimum is taken over all partitions $V(G) = V_1 \cup V_2$ with $|V_1|$, $|V_2| \geq n/3$.

Consider now a drawing of G in the plane such that the vertices of G are represented by distinct points and the edges are represented by Jordan arcs connecting them with the property that no arc passes through a vertex different from its endpoints. The crossing number $c(G)$ of G is defined as the minimum number of crossing pairs of arcs in such a drawing of G, where two arcs are said to be crossing if they have an interior point in common. It is easy to show that the minimum number of crossings can always be realized by a drawing satisfying the following conditions:

(1) no two arcs meet in more than one point (including their endpoints);

(2) no three arcs share a common interior point.

We need the following result which is an easy consequence of a weighted version of the Lipton-Tarjan separator theorem for planar graphs [LT].

Theorem 2.1 *Let G be a graph with n vertices of degree $d_1, \ldots, d_n$. Then*

$$b^2(G) \leq (1.58)^2 \left(16c(G) + \sum_{i=1}^{n} d_i^2\right),$$

where $b(G)$ and $c(G)$ denote the bisection width and the crossing number of G, respectively.

Proof: Let H be a plane graph on the vertex set $V(H) = \{v_1, \ldots, v_N\}$ such that each vertex has a non-negative weight $w(v_i)$ and $\sum_{i=1}^{N} w(v_i) = 1$. Let $d(v_i)$ denote the degree of v_i in H. It was shown by Gazit and Miller [GM] that, by the removal of at most

$$1.58 \left(\sum_{i=1}^{N} d^2(v_i)\right)^{1/2}$$

edges, H can be separated into two disjoint subgraphs H_1 and H_2 such that

$$\sum_{v_i \in V(H_1)} w(v_i) \geq \frac{1}{3} \qquad \sum_{v_i \in V(H_2)} w(v_i) \geq \frac{1}{3}.$$

(See also [M] and [DDS].)

Consider now a drawing of G with $c(G)$ crossing pairs of arcs satisfying conditions (1) and (2). Introducing a new vertex at each crossing, we obtain a plane graph H with $N = n + c(G)$ vertices. Assign weight 0 to each new vertex and weights of $1/n$ to all other vertices. The above result implies that, by the deletion of at most

$$1.58 \left(16c(G) + \sum_{i=1}^{n} d_i^2\right)^{1/2}$$

edges, H can be separated into two parts H_1 and H_2 such that both of the sets $V_1 = V(H_1) \cap V(G)$ and $V(H_2) \cap V(G)$ have at least $n/3$ elements. Hence,

$$b(G) \leq |E(V_1, V_2)| \leq 1.58 \left(16c(G) + \sum_{i=1}^{n} d_i^2\right)^{1/2},$$

and the result follows. □

In the special case when every vertex of G is of degree at most 4, Theorem 2.1 was established by Leighton [L] and it proved to be an important tool in VLSI design (see [U]).

3 Geometric graphs

The aim of this section is to prove the following generalization of Theorem 1.1 for curvilinear graphs.

Theorem 3.1 *Let G be a graph with $n \geq 2$ vertices, and let $k \geq 1$. If G has a drawing with Jordan arcs such that no arc passes through any vertex other than its edpoints and there are no $k+1$ pairwise crossing arcs, then*

$$|E(G)| \leq 3n(10\log_2 n)^{2k-2}.$$

Proof: By double induction on k and n. The assertion is true for $k=1$ and for all n. It is also true for any $k>1$ and $n \leq 6 \cdot 10^{2k-2}$, because for these values the above upper bound exceeds $\binom{n}{2}$.

Assume now that we have already proved the theorem for some k and all n, and we want to prove it for $k+1$. Let $n \geq 6 \cdot 10^{2k}$, and suppose that the theorem holds for $k+1$ and for all graphs having fewer than n vertices.

Let G be a graph of n vertices, which has a drawing with no $k+2$ pairwise crossing edges. Let us fix such a drawing with $c(G)$ crossings and, for the sake of simplicity, denote it also by $G=(V(G),E(G))$. For any arc $e \in E(G)$, let G_e denote the graph consisting of all arcs that cross e. Clearly, G_e has no $k+1$ pairwise crossing arcs. Thus, by the induction hypothesis,

$$\begin{aligned} c(G) &= \frac{1}{2}\sum_{e\in E(G)} |E(G_e)| \\ &\leq \frac{1}{2}\sum_{e\in E(G)} 3n(10\log_2 n)^{2k-2} \\ &\leq \frac{3}{2}|E(G)|n(10\log_2 n)^{2k-2}. \end{aligned}$$

Since $\sum_{i=1}^{n} d_i^2 \leq 2|E(G)|n$ holds for every graph G with degrees $d_1,\ldots,d_n$, Theorem 2.1 implies that

$$\begin{aligned} b(G) &\leq 1.58\left(16c(G)+\sum_{i=1}^{n} d_i^2\right)^{1/2} \\ &\leq 9\sqrt{n|E(G)|}\,(10\log_2 n)^{k-1}. \end{aligned}$$

Consider a partition of $V(G)$ into two parts V_1 and V_2, each containing at least n/3 vertices, such that the number of edges connecting them is $b(G)$. Let G_1 and G_2 denote the subgraphs of G induced by V_1 and V_2, respectively. Since neither of G_1 or G_2 contains $k+2$ pairwise crossing edges and each of them has fewer than n vertices, we can apply the induction hypothesis to obtain

$$\begin{aligned} |E(G)| &= |E(G_1)|+|E(G_2)|+b(G) \\ &\leq 3n_1(10\log_2 n_1)^{2k}+3n_2(10\log_2 n_2)^{2k}+b(G), \end{aligned}$$

where $n_i = |V_i|$ $(i=1,2)$. Combining the last two inequalities we get

$$\begin{aligned} &|E(G)| - 9\sqrt{n}(10\log_2 n)^{k-1}\sqrt{|E(G)|} \\ \leq\ & 3\frac{n}{3}(10\log_2\frac{n}{3})^{2k}+3\frac{2n}{3}(10\log_2\frac{2n}{3})^{2k} \\ \leq\ & 3n(10\log_2 n)^{2k}(1-\frac{k}{\log_2 n}). \end{aligned}$$

If the left hand side of this inequality is negative, then $|E(G)| \leq 3n(10\log_2 n)^{2k}$ and we are done. Otherwise,

$$f(x) = x - 9\sqrt{n}(10\log_2 n)^{k-1}\sqrt{x}$$

is a monotone increasing function of x when $x \geq |E(G)|$. An easy calculation shows that

$$f(3n(10\log_2 n)^{2k}) > 3n(10\log_2 n)^{2k}(1-\frac{k}{\log_2 n}).$$

Hence,

$$f(|E(G)|) < f(3n(10\log_2 n)^{2k}),$$

which in turn implies that

$$|E(G)| < 3n(10\log_2 n)^{2k},$$

as required. □

4 Avoiding snakes

In [ASS], Aronov, Seidel and Souvaine constructed two polygonal regions P and Q with vertices $\{p_1,\ldots,p_n\}$ and $\{q_1,\ldots,q_n\}$ in clockwise order such that the size of any piecewise linear homeomorphism $f: P \rightarrow Q$ with $f(p_i)=q_i$ $(1 \leq i \leq n)$ is at least cn^2 (for an absolute constant $c>0$). Their ingenious construction heavily relies on some special geometric features of "snakelike" polygons.

Our theorem 1.2 (stated in the introduction) provides the same lower bound for a modified version of this problem due to J.E. Goodman. The proof given below is purely combinatorial, and avoids the use of "snakes."

Proof of Theorem 1.2: Let T_1 and T_2 be two triangles containing two convex n-gons P and Q in their interiors, respectively. Let $p_{\pi(1)}, \ldots, p_{\pi(n)}$ denote the vertices of P in clockwise order, where π is a permutation of $\{1, \ldots, n\}$ to be specified later. Furthermore, let $q_1, \ldots, q_n$ denote the vertices of Q in clockwise order. Let $f : T_1 \to T_2$ be a piecewise linear homeomorphism with $f(p_i) = q_i$ $(1 \leq i \leq n)$, and fix a triangulation $\mathcal{T}_1$ of T_1 with $|\mathcal{T}_1| = \texttt{size}(f)$ triangles such that f is linear on each of them. By subdividing some members of $\mathcal{T}_1$ if necessary, we obtain a new triangulation $\mathcal{T}_1'$ of T_1 such that each p_i is a vertex of $\mathcal{T}_1'$ and $|\mathcal{T}_1'| \leq |\mathcal{T}_1| + 3n$.

Obviously, f will map $\mathcal{T}_1'$ into an isomorphic triangulation $\mathcal{T}_2'$ of T_2. The image of each segment $p_{\pi_i}p_{\pi(i+1)}$ is a polygonal path connecting $q_{\pi(i)}$ and $q_{\pi(i+1)}$, $(1 \leq i \leq n)$. The collection of these paths together with the segments q_iq_{i+1} is a drawing of the graph $G = G_\pi$ defined by:

$$V(G) = \{q_1, \ldots, q_n\},$$

(∗)

$$E(G) = \{q_iq_{i+1} \mid 1 \leq i \leq n\} \cup \{q_{\pi(i)}q_{\pi(i+1)} \mid 1 \leq i \leq n\}.$$

Suppose that this drawing has c crossing pairs of arcs. Notice that each crossing must occur between a path $q_{\pi(i)}q_{\pi(i+1)}$ and a segment q_jq_{j+1}. By the convexity of Q, any line can intersect at most two segments q_jq_{j+1}. Hence the total number of subsegments of the concatenation of the polygons $f(p_{\pi(i)}p_{\pi(i+1)})$, $1 \leq i \leq n$, is at least $c/2$. On the other hand, by the convexity of P, each triangle belonging to $\mathcal{T}_1'$ intersects at most two sides of the form $P_{\pi(i)}P_{\pi(i+1)}$. Thus, $|\mathcal{T}_1'| \geq c/4$, which yields that

$$\texttt{size}(f) = |\mathcal{T}_1| \geq |\mathcal{T}_1'| - 3n \geq \frac{c(G)}{4} - 3n,$$

where $c(G)$ stands for the crossing number of G. Applying Theorem 2.1, we obtain that

$$c(G) > \frac{b^2(G)}{40} - 1.$$

Therefore,

$$\texttt{size}(f) \geq \frac{b^2(G)}{160} - 3n - \frac{1}{4}.$$

To complete the proof of Theorem 1.2, it is sufficient to show that for a suitable permutation π the bisection width of the graph $G = G_\pi$ defined by (∗) is at least constant times n. We use a counting argument (cf. [AS]. The family of graphs G_π has size $n!$. We bound from above the number of those members of this family whose bisection width is at most k. We will see that for $k \leq n/20$ this number is less than $n!$.

Let $b(G_\pi) \leq k$. Let (V_1, V_2) be a partition of $V(G_\pi)$ with $|V_1|, |V_2| \geq n/3$ and $E(V_1, V_2) \leq k$. Define

$$\begin{aligned} E_1(V_1, V_2) &= \{q_iq_{i+1} \mid 1 \leq i < n\} \cap E(V_1, V_2), \\ E_2(V_1, V_2) &= \{q_{\pi(i)}q_{\pi(i+1)} \mid 1 \leq i < n\} \cap E(V_1, V_2). \end{aligned}$$

Since $|E_1(V_1, V_2)| \leq k$, the partition (V_1, V_2) should be of a special form. If we delete all elements of $E_1(V_1, V_2)$ from the path $q_1 \ldots q_n$, it splits into at most $k+1$ paths (or points) lying alternately in V_1 and in V_2. This yields a $2(k+1)\binom{n}{k}$ upper bound on the number of partitions in question.

The order in which the elements of V_i $(i = 1, 2)$ occur in the sequence $q_{\pi(1)} \ldots q_{\pi(n)}$ can be represented by a function $\sigma_i : \{1, \ldots, |V_i|\} \to V_i$ $(i = 1, 2)$. For a fixed partition (V_1, V_2), there are at most $|V_1|!$ choices for σ_1 and $|V_2|!$ choices for σ_2. If σ_1 and σ_2 are also fixed, then the number of possible permutations is bounded again by $2(k+1)\binom{n}{k}$. Thus the total number of permutations π for which $b(G_\pi) \leq k$ cannot exceed

$$\begin{aligned} & \sum_{(V_1, V_2)} |V_1|!|V_2|!2(k+1)\binom{n}{k} \\ \leq & \sum_{(V_1, V_2)} n!\binom{n}{n/3}^{-1} 2(k+1)\binom{n}{k} \\ \leq & \; 4(k+1)^2\binom{n}{k}^2\binom{n}{n/3}^{-1} n!, \end{aligned}$$

which is less than $n!$ if $k \leq n/20$, and n is sufficiently large. □

References

[AE] N. Alon and P. Erdős, Disjoint edges in geometric graphs, *Discrete Comput. Geom. 4 (1989), 287-290*

[AS] N. Alon and J. Spencer The Probabilistic Method, *J. Wiley, New York,1992.*

[ASS] B. Aronov, R. Seidel and D. Souvaine, On compatible triangulations of simple polygons, *Computational Geometry: Theory and Applications 3 (1993), 27-36*

[AH] S. Avital and H. Hanani, Graphs, *Gilyonot Lematematika 3 (1966), 2-8. (Hebrew)*

[CP] V. Capoyleas and J. Pach, A Turán-type theorem on chords of a convex polygon, *J. Combinat. Theory B, 56 (1992), 9-15.*

[DDS] K. Diks, H. N. Djidjev, O. Sykora and I. Vrto, Edge separators for planar graphs and their applications, *Proc. 13th Symp. Math. Found. Comp. Science (Borlin, ed), Lecture Notes in Comp. Science 324, Springer Verlag, 280-290.*

[AH] P. Erdős On sets of distances of n points, *Amer. Math. Monthly 53 (1946), 248-250.*

[GM] H. Gazit and G.L. Miller, Planar separators and the Euclidean norm, *Algorithms (Proc. Internat. Symp. SIGAL'90, T. Asano at. al. eds.), Lecture Notes in Computer Science 450, 1990, 338-347.*

[GKK] W. Goddard, M. Katchalski and D.J. Kleitman, Forcing disjoint segments in the plane, *manuscript.*

[AH] H Hopf and E. Pannwitz, Aufgabe No. 167, *Jber. Deutsch. Math, Verein. 43 (1934), 114.*

[K] Y.S. Kupitz, Extremal Problems in Combinatorial Geometry, *Aarhaus University Lecture Note Series, no. 53, Aarhaus University, Denmark, 1979.*

[L] F. T. Leighton, Complexity Issues in VLSI, Foundations of Computing Series, MIT Press, *Cambridge, Mass., 1983.*

[LT] R.J. Lipton and R.E. Tarjan, A separator theorem for planar graphs, *SIAM Journal of Applied Math. 36 (1979), 177-189.*

[M] G.L. Miller, Finding small simple cycle separators for 2-connected planar graphs, *J. Computer and System Sciences 32 (1986), 265-279.*

[OP] P. O'Donnel and M. Perles, Every geometric graph with n vertices and 3.6n-3.4 edges contains three pairwise disjoint edges, *manuscript, Rutgers University, New Brunswick, 1991.*

[P] J. Pach, Notes on geometric graph theory, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 6 (1991), 273-285.*

[PT] J Pach and J. Töröcsik, Some geometric applications of Dilworth's theorem, *Proc. 9th Annual ACM Symposium on Computational Geometry, 1993.*

[S] A. Saalfeld, Joint triangulations and triangulation maps, *Proc. 3rd Annual ACM Symposium on Computational Geometry, 1987, 195-204*

[SW] D. Souvaine and R. Wenger, Constructing piecewise linear homeomorphisms, *Submitted to 10th Annual ACM Symposium on Computational Geometry.*

[U] J.P. Ullman, Computational Aspects of VLSI, *Comp. Sci. Press, Rockville, Maryland, 1984.*

Cutting Dense Point Sets in Half[*]

Herbert Edelsbrunner[†], Pavel Valtr[‡] and Emo Welzl[§]

Abstract

A halving hyperplane *of a set S of n points in $\mathbb{R}^d$ contains d affinely independent points of S so that equally many of the points off the hyperplane lie in each of the two half-spaces. We prove bounds on the number of halving hyperplanes under the condition that the ratio of largest over smallest distance between any two points is at most $\delta n^{1/d}$, δ some constant. Such a set S is called* dense.

In $d = 2$ dimensions, the number of halving lines for a dense set can be as much as $\Omega(n \log n)$, and it cannot exceed $\mathrm{O}(n^{5/4}/\log^* n)$. The upper bound improves over the current best bound of $\mathrm{O}(n^{3/2}/\log^* n)$ which holds more generally without any density assumption. In $d = 3$ dimensions we show that $\mathrm{O}(n^{7/3})$ is an upper bound on the number of halving planes for a dense set. The proof is based on a metric argument that can be extended to $d \geq 4$ dimensions, where it leads to $\mathrm{O}(n^{d-\frac{2}{d}})$ as an upper bound for the number of halving hyperplanes.

1 Introduction

Dense point sets are sets with bounded ratio of largest over smallest distance between points. We present extremal properties of dense point sets for the number of halving hyperplanes and for the stabbing number of geometric d-uniform hypergraphs. In this section, we introduce and discuss notions and results.

Dense Sets. Let $\delta > 0$. A set S of n points in $\mathbb{R}^d$ is *δ-dense* if the ratio of the maximum over the minimum distance between two points in S is at most $\delta n^{1/d}$. We use *dense* short for δ-dense if δ is a constant. Note that values of δ that admit arbitrarily large δ-dense sets cannot be arbitrarily small. For example in $\mathbb{R}^2$, $\delta \geq \delta_0 = \sqrt{2\sqrt{3}/\pi} \approx 1.05$ is necessary and sufficient. Arbitrarily large δ_0-dense sets can be constructed as the intersection of a triangle grid with a disk. For our purposes, it is possible to suppose the minimum distance is 1, so S is δ-dense if the diameter of S is at most $\delta n^{1/d}$.

Various combinatorial extremal problems previously considered for arbitrary point sets have recently been studied for dense sets. These include convex and empty convex subsets [2, 20, 21], separation discrepancy [1], mutual avoidance and crossing families [21]. The complexity of the convex hull problem for δ-dense sets was determined in [21].

The investigation of dense point sets is motivated by the common discrepancy between the complexity of algorithms in the worst case and in practical cases. The complexity of a geometric algorithm typically depends on certain combinatorial parameters associated with the geometric data, and it is often the case that these parameters obtain their extrema only for bizarre and rare data sets. In particular, extremal point sets constructed in the literature often exhibit large distance ratios, sometimes exponential in the number of points. This is related to the fact that there are combinatorial types of point configurations that require the distance ratio be at least double-exponential in the number of points, see [14, 15]. Such sets are unlikely to occur in

[*] The research by H. Edelsbrunner is partially supported by the National Science Foundation, under grant ASC-9200301 and the Alan T. Waterman award, grant CCR-9118874. The research by P. Valtr is supported by the Deutsche Forschungsgemeinschaft, under grant We 1265/2-1. Part of the research by P. Valtr and E. Welzl has been carried out during a visit to the Tel Aviv University supported by the German-Israeli Foundation for Scientific Research and Development.

[†] Dept. Comput. Sci., Univ. Illinois at Urbana-Champaign, Urbana, Illinois 61801, U. S. A.

[‡] Graduiertenkolleg "Algorithmische Diskrete Mathematik", Freie Univ. Berlin, 14195 Berlin, Germany and Dept. Applied Math., Malostranské nám. 25, Charles Univ., 118 00 Praha 1, Czech Republic.

[§] Inst. Informatik, Freie Univ. Berlin, 14195 Berlin, Germany.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

practical applications. As a consequence, researchers spend a substantial amount of time and effort battling difficulties that are too rare to be relevant in practical applications. The notion of density can be seen as an attempt to make the theoretical analysis of algorithms more relevant to practice.

There are two related concepts. Random sets of points uniformly distributed in the unit square, and sets of points with small integer coordinates (grid points). Dense point sets behave differently from such sets, at least in some respect. For example, a random set of n points in the unit square is likely to have a pair at distance $O(\frac{1}{n})$, and is therefore not dense. Similarly, a dense set of n points in the plane can include the vertices of a convex m-gon, $m = \Omega(\sqrt{n})$, which is impossible for any set of points with integer coordinates between 0 and some constant times $\sqrt{n}$.

It is perhaps worthwhile to mention that dense point sets have been considered in computer graphics [6], and that they commonly appear in nature (eye-photoreceptor distributions, minimum distance constraints for molecules, atoms of proteins, etc.).

Halving hyperplanes. A notion that is closely related to and somewhat more general than a halving hyperplane is that of a k-set. A subset $T \subseteq S$ of a finite set $S \subseteq \mathbb{R}^d$ is a *k-set* if card $T = k$ and there is a half-space so that T is the intersection of S with this half-space. The number of k-sets, for some fixed k, arises in the analysis of geometric algorithms, see e.g. [7, 8, 12]. The number of k-sets obtains its maximum when the points are in general position, which we henceforth assume.

The problem of bounding the maximum number of k-sets possible for a set of n points turns out to be notoriously difficult, even in $d = 2$ dimensions. The largest k for which matching upper and lower bounds are known is 3, see [18]. The most difficult case seems to be when k is roughly $\frac{n}{2}$, which is the reason why this case receives special attention as the problem of bounding the number of *halving lines*. The best upper bounds for general point sets in $\mathbb{R}^2$ is $O(n^{3/2}/\log^* n)$, see [17], and there are constructions of sets with $\Omega(n \log n)$ halving lines, [11, 13]. Our results for dense sets in $\mathbb{R}^2$ consist of an adaptation of the lower bound in [11] and of the upper bound $O(n^{5/4}/\log^* n)$.

In three and higher dimensions, the problem seems even harder. The first non-trivial upper bound for $d = 3$ dimensions has been given in [5]. This bound has been improved in [4, 10], and the current best bound is $O(n^{8/3})$, as derived in [9]. In $d \geq 4$ dimensions there are minute improvements over the trivial $O(n^d)$ bound based on algebraic topology arguments about the non-embeddability of certain complexes, see [22]. We have $O(n^{d-\frac{2}{d}})$ as an upper bound for dense sets in all $d \geq 3$ dimensions.

Outline. Section 2 shows the $\Omega(n \log n)$ lower bound on the number of halving lines is asymptotically unaffected by the density assumption. Section 3 gives a proof of the $O(n^{\frac{5}{4}}/\log^* n)$ upper bound on the number of halving lines for a dense set of n points in $\mathbb{R}^2$. The crux of the argument is a reduction to k-sets for about $\sqrt{n}$ subsets of size at most about $\sqrt{n}$ each. Section 4 considers point sets in $\mathbb{R}^3$. The approach to proving an upper bound rests on an area argument combined with an observation about centroids of subsets of the set. This is extended to $d \geq 4$ dimensions in section 5. Alternatively, the metric argument can be combined with an upper bound on the stabbing number of geometric d-uniform hypergraphs known as Lovász' lemma [16]. This is discussed in section 6.

2 Lower bound in the plane

The constructions in [11, 13] establishing the $\Omega(n \log n)$ lower bound on the number of halving lines are not dense: with minimum distance 1 they require the maximum distance be at least some constant times n. This section shows that asymptotically the same lower bound can also be obtained for dense sets. We begin by modifying the construction in [11] so that the x_1-coordinates of the points are contiguous integers.

LEMMA. For any positive even integer m there are points $p_1, p_2, \ldots, p_m$ in $\mathbb{R}^2$ so that

(i) for all i the x_1-coordinate of p_i is i, and

(ii) there are at least $\frac{m}{6} \log_3 \frac{m}{2}$ halving lines.

PROOF. Assume first that $m = 2 \cdot 3^k$, $k \geq 0$, and denote the number of halving lines of a set S by $h(S)$. We construct sets S_k, card $(S_k) = 2 \cdot 3^k$, with slightly higher number of halving lines than claimed, namely $h(S_k) = (k+1) \cdot 3^k = \frac{m}{2} \log_3 \frac{3m}{2}$.

For $k = 0$ we have $m = 2$, $S_0 = \{(1,0),(2,0)\}$, and $h(S_0) = 1$. We continue by induction. S_{k+1} is the union of three sets, T_0, T_1, T_2, each an affinely transformed copy of S_k as described below. By horizontal translation, along the x_1-axis, make sure the points in T_i have x_1-coordinates from $im + 1$ through $(i+1)m$. Next, shrink the x_2-coordinates of all points in T_i by a sufficiently large factor. Then, move the points vertically, along the x_2-axis, so that the shrunk x_2-coordinates become vertical displacements with respect to a line, one for each T_i. The three lines meet in point $(\frac{2m+1}{2}, 0)$ and have slopes 0, 1, and -1. The line for T_0 is $x_2 = 0$, the one for T_1 is $x_2 = x_1 - \frac{2m+1}{2}$, and the one for T_2 is $x_2 = -x_1 + \frac{2m+1}{2}$. So in fact, the only difference between S_k and T_0 is in the x_2-coordinates which are scaled down for T_0; the points in T_1 and T_2 also experience horizontal translation and vertical displacement other than scaling. The shrinking factor for each set T_i is chosen sufficiently large so that any line through two points of T_i separates T_{i+1} and T_{i+2}, where indices are taken modulo 3.

Any halving line of T_i is also a halving line of S_{k+1}. In addition, S_{k+1} has $\frac{m}{2}$ halving lines connecting a point of T_i with a point of T_{i+1} each, for $i = 0, 1, 2$. It follows that

$$\begin{aligned} h(S_{k+1}) &= h(T_0) + h(T_1) + h(T_2) + 3m/2 \\ &= 3h(S_k) + 3^{k+1} \\ &= (k+2)3^{k+1}. \end{aligned}$$

To extend this construction to arbitrary positive even m observe that all halving lines of S_{k+1} have bounded slope. We can thus add equally many points above and below all these lines and retain them as halving lines. To satisfy the integer coordinate requirement, we add the points with contiguous integer x_1-coordinates, starting at $2 \cdot 3^k + 1$. After adding j points above and j points below all halving lines of S_k, we get a set of $m = 2 \cdot 3^k + 2j$ points, which we denote by P_m for later reference. We may assume $j < 2 \cdot 3^k$, so P_m has fewer than three times as many points than S_k, which implies the claimed lower bound for $h(P_m)$. □

The integer x_1-coordinates can be used to control the density of transformed copies of P_m. This is done in the lower bound construction below.

THM. 1 For any even $n \geq 2$ there is a 2-dense set of n points in $\mathbb{R}^2$ with at least $\frac{n}{12} \log_3 n - n$ halving lines.

PROOF. We may assume that $n \geq 3^{12}$, for otherwise the claim is void. Define $m = 2 \cdot \lfloor \frac{\sqrt{n}}{4} \rfloor$ and $M = 2 \cdot \lceil \sqrt{n} \rceil + 9$; so m is even and M is odd. Let Y be the annulus of points on and between the two circles with radii m and $2m$, both centered at the origin. For $1 \leq i \leq M$, let R_i be the half-line with angle $\frac{i-1}{M} \cdot 2\pi$ starting at the origin. The intersection $R_i \cap Y$ is a line segment of length m. For $\varepsilon > 0$ and $1 \leq i \leq M$ find an affine transformation Q_i of P_m so that

(i) $Q_i \subseteq Y$,

(ii) the distance of any $p \in Q_i$ from R_i is at most ε, and

(iii) the distance between any two points in Q_i is at least 1.

For example, shrink the x_2-coordinates of the points in P_m to within $[-\varepsilon, \varepsilon]$, rotate the set by $\frac{i-1}{M} \cdot 2\pi$ about the origin, and translate along R_i until all points lie in Y. Define $S = \bigcup_{i=1}^{M} Q_i$. For sufficiently small $\varepsilon > 0$, every halving line of Q_i is also a halving line of S.

Finally, we adjust the number of points from $\operatorname{card} S = M \cdot m = (2\lceil \sqrt{n} \rceil + 9)(2\lfloor \frac{\sqrt{n}}{4} \rfloor)$ to n. Notice that

$$n < n + \frac{\sqrt{n}}{2} - 18 \leq \operatorname{card} S \leq n + \frac{11\sqrt{n}}{2} < n + 12m.$$

We can delete points from 12 sets Q_i until a set $S' \subseteq S$ with $\operatorname{card} S' = n$ remains. By choosing half these points from Q_j, $1 \leq j \leq 6$, and the other half from Q_j, $\frac{M+1}{2} \leq j \leq \frac{M+11}{2}$, the halving lines of the remaining sets Q_i are still halving lines of S'. Therefore,

$$\begin{aligned} h(S') &\geq (M-12) \cdot h(P_m) \\ &\geq (2\sqrt{n} - 3) \cdot \frac{\sqrt{n} - 4}{12} \log_3 \frac{\sqrt{n} - 4}{4} \\ &\geq \frac{n}{12} \log_3 n - n. \end{aligned}$$

The diameter of S' is less than $4m$ and the number of points is $n \geq 4m^2$. The density of S' is therefore $\delta < \frac{4m}{\sqrt{4m^2}} = 2$. □

REMARK. It is possible to improve the density factor by setting $m \approx \sqrt{\sqrt{3}/(4\pi)} \cdot \sqrt{n} \approx 0.37\sqrt{n}$ and $M \approx \frac{2}{\sqrt{3}} \cdot 2\pi m \approx \sqrt{4\pi/\sqrt{3}} \cdot \sqrt{n} \approx 2.69\sqrt{n}$ and by placing the m points of Q_i alternately at distances roughly $m + \frac{1}{4}$, $m + 1 + \frac{1}{4}$, $\cdots$, $2m - 1 + \frac{1}{4}$ and at distances roughly $m + \frac{3}{4}$, $m + 1 + \frac{3}{4}$, $\cdots$, $2m - 1 + \frac{3}{4}$ from the origin. The diameter is smaller than $4m$, so the density is at most $2\sqrt{\sqrt{3}/\pi} \approx 1.49$. The number of halving lines is at least $\frac{n}{12} \log_3 n - n$.

For any fixed $\delta > \sqrt{2\sqrt{3}/\pi} \approx 1.05$, we can construct a δ-dense set of even size n with at least $cn \log n - n$ halving lines, where $c = c(\delta) > 0$ is independent of n, as follows. Put $m_1 = \frac{1}{2}\delta\sqrt{n}$ and $m_2 = (\frac{1}{2}\delta - \varepsilon)\sqrt{n}$, where $\varepsilon = \varepsilon(\delta) > 0$ is sufficiently small. Let Y be the annulus of points on and between the two circles with radii m_1 and m_2, both centered at the origin. We find a set S of $4m_2(m_1 - m_2)$ points in the annulus Y similarly as in the proof of Theorem 1 so that S has at least $cn \log n - n$ halving lines. Let T be a set of points of a triangle grid with minimum distance 1 which is symmetric around the origin and does not contain the origin. Let T' be the intersection of T with the disk of radius $m_2 - 1$ centered at the origin. We may certainly assume that each of the $cn \log n - n$ halving lines of S cuts T' into two equal-sized parts. Then, $S \cup T'$ has at least $cn \log n - n$ halving lines. If $\varepsilon > 0$ was chosen sufficiently small, $S \cup T'$ has at least n points. Finally, we delete $(\operatorname{card}(S \cup T') - n)/2$ pairs of points of T', each pair symmetric around the origin, so that exactly n points remain in $S \cup T'$. This gives us a δ-dense set of size n with at least $cn \log n - n$ halving lines.

3 Upper bound in the plane

Let $f_k(n)$ be the maximum number of k-sets in a set of n points in the plane, and let $f(n) = \max\{f_k(n) : 0 \leq k \leq n\}$. As mentioned in the introduction, $f(n) = \mathrm{O}(n^{\frac{3}{2}}/\log^* n)$ is the best known upper bound on $f(n)$. We show that δ-dense sets admit smaller bounds. In particular, the result implies that $\mathrm{O}(n^{\frac{5}{4}}/\log^* n)$ is an upper bound for dense sets.

THM. 2 The number of halving lines of a δ-dense set of n points in $\mathbb{R}^2$ is at most $\mathrm{O}(\delta\sqrt{n} \cdot f(3\delta\sqrt{n}))$.

PROOF. Let S be a δ-dense set of n points in $\mathbb{R}^2$. Consider a general direction $\gamma \in [0, \pi)$ and a line $\ell = \ell(\gamma)$ with direction γ so that equally many points of S lie on both side of ℓ. The length of $\ell \cap \operatorname{conv} S$ is at most the diameter of S, which is at most $\delta\sqrt{n}$. Let $R = R(\gamma)$ be the smallest rectangle that contains the $\frac{1}{2}$-neighborhood of $\ell \cap \operatorname{conv} S$; its length is at most $\delta\sqrt{n} + 1$, its width is 1, so its area is at most $\delta\sqrt{n} + 1$. For each point $p \in S \cap R$ consider the disk $D_p = \{x \mid |xp| < \frac{1}{2}\}$, where $|xp|$ is the Euclidean distance between x and p. The area of R covered by D_p is at least $\frac{\pi}{8}$. Since all such disks are disjoint, this implies that

$$\operatorname{card}(S \cap R) \leq \frac{\delta\sqrt{n}+1}{\frac{\pi}{8}} < \frac{8}{\pi} \cdot \delta\sqrt{n} + 3.$$

We say a line h meets R *shortside* if h meets the two short sides, which have length 1 each. A halving line of S necessarily meets ℓ within $\operatorname{conv} S$ and thus within R. It follows that if h is a halving line then it meets R shortside if its direction differs from γ by at most x, where $(\delta\sqrt{n} + \frac{1}{2}) \tan x = \frac{1}{2}$, or equivalently, $x = \arctan \frac{1}{2 \cdot \delta\sqrt{n}+1}$. Since $\delta\sqrt{n}$ certainly exceeds 1, we have

$$\arctan \frac{1}{2\delta\sqrt{n}+1} \geq \frac{1}{2\delta\sqrt{n}+1} \geq \frac{1}{3 \cdot \delta\sqrt{n}} > \frac{\pi}{10 \cdot \delta\sqrt{n}}.$$

Choose $m = \lceil 5 \cdot \delta\sqrt{n} \rceil$ and assume that all directions $\gamma = 0, \frac{1}{m}\pi, \frac{2}{m}\pi, \ldots, \frac{m-1}{m}\pi$ are general in the sense that no two points of S lie on a line with such a direction. Two contiguous directions differ by less than $2x$, so each halving line meets at least one of the rectangles $R(\gamma)$ shortside.

To finish the argument observe that each rectangle, R, is met shortside by at most $f(\frac{8}{\pi}\delta\sqrt{n} + 3)$ halving lines, because they are all split $S \cap R$ the same way. It follows that the total number of halving lines is at most

$$m \cdot f(\frac{8}{\pi}\delta\sqrt{n} + 3) = \mathrm{O}(\delta\sqrt{n} \cdot f(3\delta\sqrt{n})),$$

as claimed. □

REMARK. Let $f_k(n, \delta)$ be the maximum number of k-sets in a δ-dense set of n points in the plane, and let $f(n, \delta) = \max\{f_k(n, \delta) : 0 \leq k \leq n\}$. Thus, Theorem 2 yields

$$f_k(n, \delta) = \mathrm{O}(\delta\sqrt{n} \cdot f(3\delta\sqrt{n})),$$

where $k = \frac{n}{2} - 1$. An analogous argument gives also

$$f(n, \delta) = \mathrm{O}(\delta\sqrt{n} \cdot f(3\delta\sqrt{n})).$$

REMARK. Let us redefine $R = R(\gamma)$ in the proof of Theorem 2 to be the smallest rectangle that contains the ε-neighborhood of $\ell \cap \operatorname{conv} S$, where $\varepsilon > 0$ is a small positive constant. In this case, the points of $S \cap R$ are almost collinear. We call $S \subseteq \mathbb{R}^2$ of size n *δ-line-dense* if there is a line q such that the orthogonal projection of S to q is a δ-dense set in the 1-dimensional interpretation.

Let $l_k(n, \delta)$ be the maximum number of k-sets in a δ-line-dense set of n points in the plane, and let $l(n, \delta) = \max\{l_k(n, \delta) : 0 \leq k \leq n\}$. Analogous arguments as in sections 2 and 3 yield that for any fixed $\delta > \delta_0 = \sqrt{2\sqrt{3}/\pi}$ and $\delta' > 1$,

$$f(n, \delta) = \mathrm{O}(\sqrt{n} \cdot l(c_1\sqrt{n}, \delta')) = \mathrm{O}(f(c_2 n, \delta)),$$

where c_1 and c_2 are two constants depending only on δ and δ'.

In particular, if the function $f(n, \delta)$ is smooth as a function of n (i.e., $f(cn, \delta) = \mathrm{O}(f(n, \delta))$ for any constant c) then

$$f(n, \delta) = \Theta(\sqrt{n} \cdot l(\sqrt{n}, \delta')),$$

and determining the asymptotic behavior of the maximum number of halving lines is equally difficult for dense sets as for line-dense sets. We believe that the function $f(n, \delta)$ (and also the function $f(n)$) is smooth but have no proof at this moment.

REMARK. Note that Thm. 2 can be bootstrapped. So if dense sets asymptotically maximize the number of halving lines, then this maximum is $\mathrm{O}(n \text{ polylog } n)$.

4 Upper bound in space

Let S be a dense set of n points in $\mathbb{R}^3$. As mentioned in the introduction, we derive an upper bound on the number of halving planes using an area argument for triangles spanned by points in S. In particular, we need a lower bound on the total area of a collection of m such triangles. We begin with two lemmas. The collection of subsets of S with size k is denoted by $\binom{S}{k}$.

LEMMA A. Let S be a set of n points with minimum distance 1 in $\mathbb{R}^3$. Then

$$\sum_{\{p,q\}\in\binom{S}{2}} \frac{1}{|pq|} \le 3 \cdot n^{\frac{5}{3}}.$$

PROOF. Consider the ordered sequence of the $N = \binom{n}{2}$ distances: $x_1 \le x_2 \le \ldots \le x_N$. The pigeon-hole principle implies that for each index i there is a point with at least $\frac{2i}{n}$ points at distance x_i or less. So we have at least $\frac{2i}{n} + 1$ points inside a ball of radius x_i. Because of the minimum distance assumptions we get at least $\frac{2i}{n} + 1$ pairwise disjoint open balls of radius $\frac{1}{2}$ each inside a ball of radius $x_i + \frac{1}{2}$. By dividing volumes we get

$$\frac{2i}{n} + 1 \le \frac{(x_i + \frac{1}{2})^3}{(\frac{1}{2})^3} = 8(x_i + \frac{1}{2})^3 \le 3^3 \cdot x_i^3.$$

It follows that $x_i > \frac{1}{3}(\frac{i}{n})^{\frac{1}{3}}$, and therefore

$$\sum_{\{p,q\}\in\binom{S}{2}} \frac{1}{|pq|} = \sum_{i=1}^{N} \frac{1}{x_i} < 3n^{\frac{1}{3}} \sum_{i=1}^{N} i^{-\frac{1}{3}} < 3 \cdot n^{2-\frac{1}{3}}.$$

□

Based on the bound for distances we can now prove a bound for areas of triangles.

LEMMA B. Let S be a set of n points with minimum distance 1 and maximum distance D in $\mathbb{R}^3$. Then the total area of $m > 12Dn^2$ different triangles is at least $\frac{1}{40}m^{\frac{3}{2}}/n^{\frac{5}{6}}$.

PROOF. For two points $p, q \in S$ consider all third points $r \in S$ so that the area of the triangle pqr is at most A and pq is its longest edge. The distance from r to the line through p and q is therefore at most $\frac{2A}{|pq|}$, and r lies between the two planes through p and q normal to pq. The number of points r that can possibly lie inside this cylinder can be bounded by a packing argument. After growing by $\frac{1}{2}$ in every direction, the volume of the cylinder is

$$V = (|pq| + 1)(\frac{2A}{|pq|} + \frac{1}{2})^2 \pi \le \frac{8\pi A^2}{|pq|} + 4\pi A + \frac{\pi |pq|}{2},$$

because $|pq| + 1 \le 2|pq|$. It contains at most $\frac{6}{\pi}V$ pairwise disjoint open balls of radius $\frac{1}{2}$ each. Using lemma A we get an upper bound on the number of triangles with area A or less:

$$\sum_{\{p,q\}\in\binom{S}{2}} \frac{6}{\pi}V < 144A^2 n^{\frac{5}{3}} + 12An^2 + \frac{3D}{2}n^2.$$

If $A_1 \le A_2 \le \ldots \le A_M$ is the ordered sequence of the $M = \binom{n}{3}$ triangle areas, we can rewrite this as

$$i \le 144A_i^2 n^{\frac{5}{3}} + 12A_i n^2 + \frac{3D}{2}n^2.$$

For $i > 6Dn^2$ this implies that at least one of $144A_i^2 n^{\frac{5}{3}}$ and $12A_i n^2$ exceeds $\frac{1}{2}(i - \frac{3D}{2}n^2) > \frac{3i}{4}$. The first inequality implies

$$A_i > \frac{i^{\frac{1}{2}}}{14n^{\frac{5}{6}}},$$

which is weaker than the second inequality provided $i > \frac{8}{7}n^{\frac{7}{3}}$. This is indeed the case if $i > 6Dn^2$, because $D > \frac{1}{2}n^{\frac{1}{3}}$, which can be proved by a straightforward packing argument. Finally, the total area of $m = 2i > 12Dn^2$ triangles is at least

$$i \cdot A_i > \frac{m}{2} \cdot \frac{(\frac{m}{2})^{\frac{1}{2}}}{14 \cdot n^{\frac{5}{6}}} > \frac{1}{40} \cdot \frac{m^{\frac{3}{2}}}{n^{\frac{5}{6}}}.$$

□

The lower bound on the total area of m triangles is sufficient to prove an upper bound on the number of halving planes of a δ-dense set S. Assume n is odd and define $k = \frac{n-1}{2}$. For every subset $T \subseteq S$ of size k, let γ_T be the centroid, that is, $\gamma_T = \frac{1}{k}\sum_{p\in T} p$. Let S_k be the set of k-fold centroids, that is, $S_k = \{\gamma_T \mid T \subseteq S, \operatorname{card} T = k\}$. The convex hull of S_k is a three-dimensional convex polytope, P_k. Note that P_k is contained in the convex hull of S, and by density assumption, its projections to the three coordinate planes have area at most $\delta^2 n^{\frac{2}{3}}$ each. It follows that the surface area of P_k is less than $6\delta^2 n^{\frac{2}{3}}$.

We claim that if the plane, h, through points $p, q, r \in S$ is a halving plane then a homothetic copy of the triangle pqr is a face of P_k. Let $U \subseteq S$ be the subset of points on one side of h, so $\operatorname{card} U = k - 1$. For each $x \in \{p, q, r\}$, there is a plane that separates $U \cup \{x\}$ from $S - U - \{x\}$, and therefore $\gamma_{U\cup\{x\}}$ is a vertex of P_k. Furthermore, the centroids defined by $U \cup \{p\}$, $U \cup \{q\}$, and $U \cup \{r\}$ span a triangular face of P_k. This face is a homothetic copy of pqr, with scaling factor $\frac{1}{k}$. In summary, if the plane through points p, q, r is a halving plane then a homothetic copy of pqr is a face of P_k, and the area of pqr is k^2 times the area of the corresponding facet. Since the surface area of P_k is less than $6\delta^2 n^{2/3}$, the total area of all triangles corresponding to halving planes is less than $6k^2\delta^2 n^{2/3} < \frac{3}{2}\delta^2 n^{8/3}$. Lemma B thus implies the following upper bound on the number of halving planes.

THM. 3 The number of halving planes of an δ-dense set of n points in $\mathbb{R}^3$ is less than $20 \cdot \delta^{4/3} n^{7/3}$.

5 Four and higher dimensions

The area argument of section 4 can be extended to $\mathbb{R}^d$, $d \geq 4$. To simplify the argument, only dense sets of n points in $\mathbb{R}^d$ are considered, so the diameter is $D \leq \delta n^{\frac{1}{d}}$ for some constant δ. Suitable positive constants are used throughout this section and denoted as c, with or without sub- and superscript. A set T of $\operatorname{card} T = k + 1 \leq d+1$ points in $\mathbb{R}^d$ spans a k-simplex, $\sigma_T = \operatorname{conv} T$. Its k-dimensional Euclidean measure is denoted by $|\sigma_T|$. We let $s_k(A)$ denote the number of k-simplices spanned by the points whose measure is A or less. All logarithms are to the base 2.

LEMMA C. Let S be a dense set of n points in $\mathbb{R}^d$. Then $s_k(A)$ is bounded from above by

$$\begin{cases} c \cdot nA^d & \text{for } k = 1, \\ c \cdot n^{1+\frac{\binom{k}{2}}{d}} A^d \log A + c' n^{k+\frac{k-1}{d}} & \text{for } 2 \leq k \leq d. \end{cases}$$

PROOF. Let F be an $(i-1)$-flat, $1 \leq i \leq d$. The $(i-1)$-dimensional measure of its intersection with $\operatorname{conv} S$ is less than D^{i-1}. It follows that the d-dimensional measure of the cylinder of points at distance $z + \frac{1}{2}$ of less from F is less than $c_1(z+\frac{1}{2})^{d-i+1}D^{i-1}$. With a different constant, c'_1, this is also an upper bound on the number of points in S at distance z or less from F.

Consider an arbitrary ordering, $p_0, p_1, \ldots, p_k$, of the vertices of a k-simplex σ_T, $T \subseteq S$. For $1 \leq i \leq k$, let x_i be the distance of p_i from the $(i-1)$-flat spanned by $p_0, \ldots, p_{i-1}$. If $z_1, z_2, \ldots, z_k$ is a sequence of distances so that $x_i \leq z_i$ for all i, then

$$|\sigma_T| = \frac{1}{k!}\prod_{i=1}^{k} x_i \leq \frac{1}{k!}\prod_{i=1}^{k} z_i. \tag{1}$$

The number of k-simplices for which (1) holds is bounded from above by

$$c_2 n \prod_{i=1}^{k}[(z_i + \frac{1}{2})^{d-i+1}D^{i-1}].$$

The number of k-simplices with at least one x_i at most 1 is less than $c' \cdot n^k \cdot D^{k-1}$. For the others we can assume all z_i exceed 1, so the upper bound can be simplified to

$$c'_2 n D^{\binom{k}{2}} \prod_{i=1}^{k} z_i^{d-i+1}. \tag{2}$$

For a given A, we can now derive an upper bound on $s_k(A)$, the number of k-simplices σ_T with $|\sigma_T| \leq A$. For $k = 1$ set $z_1 = A$, get $|\sigma_T| = x_1 \leq A$ from (1) and $s_1(A) \leq c'_2 nA^d$ from (2). This completes the proof of the first inequality in lemma C. For $k \geq 2$ we consider all sequences z_1 though z_k so that $z_i > 1$, for all i, and $\prod_{i=1}^{k} = k!A$. Simplices are counted only if all their x_i's exceed 1; the others are covered by the second term on the right side of the second inequality in lemma C. To consider all sequences we use induction and the fact that $\prod_{i=1}^{k-1} z_i = k!\frac{A}{z_k}$. So

$$s_k(A) < \int_{z=1}^{A} s_{k-1}(\frac{A}{z}) \cdot z^{d-k+1} \cdot D^{k-1} dz.$$

For $k = 2$ we get

$$\begin{aligned} s_2(A) &< c \cdot nA^d \cdot D \int_{z=1}^{A} \frac{1}{z} dz \\ &< c''_2 \cdot nA^d \log A \cdot D, \end{aligned}$$

and for $k \geq 3$ we get

$$\begin{aligned} s_k(A) &< c \cdot nA^d \log A \cdot D^{\binom{k}{2}} \int_{z=1}^{A} \frac{1}{z^{k-1}\log z} dz \\ &< c'''_2 \cdot nA^d \log A \cdot D^{\binom{k}{2}}. \end{aligned}$$

$\square$

The upper bound on $s_k(A)$ can be used to prove a lower bound on the total measure of a collection of k-simplices, all spanned by S.

LEMMA D. Let S be a dense set of n points in $\mathbb{R}^d$. The total k-dimensional measure of $m > c'' \cdot n^{k+\frac{k-1}{d}}$ k-simplices is at least

$$\begin{cases} m^{1+\frac{1}{d}}/(c \cdot n^{\frac{1}{d}}) & \text{for } k = 1 \\ m^{1+\frac{1}{d}}/(c \cdot n^{\frac{2d+k(k-1)}{2d^2}} \log^{\frac{1}{d}} m) & \text{for } 2 \leq k \leq d. \end{cases}$$

PROOF. Let $A_1 \leq A_2 \leq \ldots \leq A_M$ be the ordered sequence of the $M = \binom{n}{k+1}$ k-simplex measures. Assume $c'' > 4c'$ so the majority of any set of m k-simplices have all x_i exceeding 1, see the proof of lemma C. For $k = 1$, we can rewrite the result in lemma C as $i < c \cdot nA_i^d$, so

$$\sum_{i=1}^{m} A_i > \frac{1}{(cn)^{\frac{1}{d}}} \sum_{i=1}^{m} i^{\frac{1}{d}} > c_3 \cdot \frac{m^{1+\frac{1}{d}}}{n^{\frac{1}{d}}}.$$

For $k \geq 2$ and $i > \frac{m}{2} > 2c'n^{k+\frac{k-1}{d}}$, we get $i < 2c \cdot n^{\ell} A_i^d \log A_i$, where $\ell = 1 + \binom{k}{2}/d$. Thus

$$A_i > c_4 \cdot \frac{(i \cdot n^{-\ell})^{\frac{1}{d}}}{\log^{\frac{1}{d}}(i \cdot n^{-\ell})}.$$

For the total measure we get

$$\begin{aligned} \sum_{i=1}^{m} A_i &> c_4 \cdot \frac{1}{n^{\frac{\ell}{d}}} \sum_{i=2}^{m} \frac{i^{\frac{1}{d}}}{\log^{\frac{1}{d}} i} \\ &> c'_4 \cdot \frac{m^{1+\frac{1}{d}}}{n^{\frac{\ell}{d}} \log^{\frac{1}{d}} m}, \end{aligned}$$

because $\int_{x=2}^{m} \frac{x^a}{\log^a x} dx < c_4'' \cdot \frac{m^{1+a}}{\log^a m}$. □

REMARK. For $d = 3$, $k = 2$, and D proportional to $n^{\frac{1}{3}}$, the bounds in lemmas B and D agree up to the logarithmic factor. The larger D gets the further the bound in lemma D lags behind the one in lemma B.

For $k = d - 1$, lemma D says that $m > c'' \cdot n^{d-\frac{2}{d}}$ $(d-1)$-simplices have total $(d-1)$-dimensional measure at least

$$\frac{m^{1+\frac{1}{d}}}{c \cdot n^{\frac{d^2-d+2}{2d^2}} \log^{\frac{1}{d}} m} > \frac{c_5}{\log^{\frac{1}{d}} n} \cdot n^{d+\frac{1}{2}-\frac{3}{2d}-\frac{3}{d^2}}.$$

The exponent of n exceeds $d-\frac{1}{d}$ for $d \geq 4$. This is worth noticing because the same argument on centroids which leads to theorem 3 in section 4 implies that

$$c_6 D^{d-1} n^{d-1} < c_6' n^{d-\frac{1}{d}}$$

is an upper bound on the total measure of all $(d-1)$-simplices spanning halving hyperplanes. It follows that there cannot be $m > c'' \cdot n^{d-\frac{2}{d}}$ such $(d-1)$-simplices, because this would contradict lemma D.

THM. 4 The number of halving hyperplanes of a dense set of n points in $\mathbb{R}^d$, $d \geq 4$, is less than $c \cdot n^{d-\frac{2}{d}}$.

6 Stabbing number

As described by Santaló [19], there is a motion-invariant measure on lines in $\mathbb{R}^3$ with the property that the measure of lines intersecting a triangle is twice its area. This measure can be used to extend theorem 3 to the stabbing problem for triangles defined by a δ-dense set $S \subseteq \mathbb{R}^3$. The measure of lines intersecting the convex hull of S is bounded by $O(\delta^2 n^{2/3})$. If we take every line with the multiplicity of its stabbing number in a set of m triangles defined by S, then this measure is twice the total area of these triangles. Lemma B shows that this area is $\Omega(m^{3/2}/n^{5/6})$, provided $m > c\delta n^{7/3}$. So if the set of triangles has the property that every line stabs at most $O(n^2)$ of them, then $m = O(\delta^{4/3} n^{7/3})$, which is the same bound as in theorem 3. Lovász' lemma guarantees that this is indeed the case for triangles that span halving planes. Thus, the motion-invariant measure for lines together with Lovász' lemma gives another proof of theorem 3 based on the area bound in lemma B. The same can be said about $d \geq 4$ dimensions, theorem 4, and lemma D.

References

[1] R. Alexander. Geometric methods in the study of irregularities of distribution. *Combinatorica* **10** (1990), 115–136.

[2] N. Alon, M. Katchalski and W. R. Pulleyblank. The maximum size of a convex polygon in a restricted set of points in the plane. *Discrete Comput. Geom.* **4** (1989), 245–251.

[3] N. Alon and M. A. Perles. On the intersection of edges of a geometric graph by straight lines. *Discrete Math.* **60** (1986), 75–90.

[4] B. Aronov, B. Chazelle, H. Edelsbrunner, L. J. Guibas, M. Sharir and R. Wenger. Points and triangles in the plane and halving planes in space. *Discrete Comput. Geom.* **6** (1991), 435–442.

[5] I. Bárány, Z. Füredi and L. Lovász. On the number of halving planes. *In* "Proc. 5th Ann. Sympos. Comput. Geom., 1989", 140–144.

[6] R. L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graphics* **5** (1986), 51–72.

[7] B. Chazelle and F. P. Preparata. Halfspace range search: an algorithmic application of k-sets. *Discrete Comput. Geom.* **1** (1986), 83–93.

[8] R. Cole, M. Sharir and C. K. Yap. On k-hulls and related problems. *SIAM J. Comput.* **16** (1987), 61–77.

[9] T. K. Dey and H. Edelsbrunner. Counting triangle crossings and halving planes. *In* "Proc. 9th Ann. Sympos. Comput. Geom., 1993", 270–273.

[10] D. Eppstein. Improved bounds for intersecting triangles and halving planes. Rept. 91-60, Dept. Comput. Sci., Univ. California at Irvine, 1991.

[11] H. Edelsbrunner and E. Welzl. On the number of line separations of a finite set in the plane. *J. Combin. Theory, Ser. A* **38** (1985), 15–29.

[12] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM J. Comput.* **15** (1986), 271–284.

[13] P. Erdős, L. Lovász, A. Simmons and E. G. Straus. Dissection graphs of planar point sets. In *A Survey of Combinatorial Theory*, ed. J. N. Srivastava et al., 139–149, North-Holland, Amsterdam, 1973.

[14] J. E. Goodman, R. Pollack and B. Sturmfels. Coordinate representation of order types requires exponential storage. *In* "Proc. 21st Ann. ACM Sympos. Theory Comput., 1989, 405–410.

[15] J. E. Goodman, R. Pollack and B. Sturmfels. The intrinsic spread of a configuration in $\mathbb{R}^d$. *J. Amer. Math. Soc.* **3** (1990), 639–651.

[16] L. Lovász. On the number of halving lines. *Ann. Univ. Sci. Budapest Eötvös Sect. Math.* **14** (1971), 107–108.

[17] J. Pach, W. Steiger and E. Szemerédi. An upper bound on the number of planar k-sets. *Discrete Comput. Geom.* **7** (1992), 109–123.

[18] E. Ramos. The number of 3-sets of a finite point set in the plane. Rept. UIUCDCS-R-93-1843, Dept. Comput. Sci., Univ. Illinois at Urbana-Champaign, Illinois, 1993.

[19] L. A. Santaló. *Integral Geometry and Geometric Probability.* Addison-Wesley, Reading, Mass., 1976.

[20] P. Valtr. Convex independent sets and 7-holes in restricted planar point sets. *Discrete Comput. Geom.* **7** (1992), 135–152.

[21] P. Valtr. Planar point sets with bounded ratios of distances. Ph.D. thesis, Fachbereich Math., Freie Univ. Berlin, 1994.

[22] R. T. Živaljević and S. T. Vrećica. The colored Tverberg's problem and complexes of injective functions. *J. Combin. Theory, Ser. A* **61** (1992), 309–318.

Fast Greedy Triangulation Algorithms*

(Extended Abstract)

Matthew T. Dickerson†
Department of Mathematics and Computer Science
Middlebury College, Middlebury VT 05753

Robert L. Scot Drysdale‡
Scott A. McElfresh
Department of Mathematics and Computer Science
Dartmouth College, Hanover NH 03755

Emo Welzl
Institut für Informatik
Freie Universität Berlin

1 Introduction

1.1 Overview of the Results

The greedy triangulation (GT) of a set S of n points in the plane is the triangulation obtained by starting with the empty set and at each step adding the shortest compatible edge between two of the points, where a compatible edge is defined to be an edge that crosses none of the previously added edges. In this paper we present a simple, practical algorithm that computes the greedy triangulation in expected time $O(n \log n)$ and space $O(n)$ for points uniformly distributed over any convex shape. A variant of this algorithm should also be fast for many other distributions.

We first describe a surprisingly simple method for testing the compatibility of a candidate edge with edges in a partially constructed greedy triangulation. Our method requires $\Theta(1)$ time for both the edge test and the update of our data structure, $\Theta(n)$ time for initialization, and $\Theta(n)$ space.

We next prove that an edge cannot be greedy if a small disk centered at its midpoint contains a point from S in both half-disks. Based on this, we are able to prove that all edges in a greedy triangulation of uniformly distributed points over a convex compact region $\mathcal{C}$ are expected either to be short or to have both endpoints near the boundary of $\mathcal{C}$.

These results allow us to develop an algorithm that computes the greedy triangulation for uniformly distributed points in $O(n \log n)$ expected time. We also present another two-phase algorithm that is less tuned to uniform distributions. It runs in expected time $O(n \log^2 n)$ time on uniformly distributed points, and $O(n^2 \log n)$ in the worst case. These algorithms should be compared to an algorithm by Levcopolous and Lingas [20]. For the more restricted case of points uniformly distributed in the unit square, their algorithm runs in expected time $O(n)$. Extending it to rectangles is straightforward, but extending it to non-rectangular convex shapes would require non-trivial modications of the algorithm and analysis. Although their algorithm is beautiful theoretically, it has not been implemented and would be difficult to implement practically. Our algorithms not only work efficiently for more general compact convex regions, but the simplicity of our algorithms and smaller constant factors would make them preferable for most practical-sized problems.

*This paper benefitted from discussions with David Eppstein and others during the *MSI Workshop on Computational Geometry,* Stony Brook NY, October 23-24, 1992, with Cliff Stein and others in the Dartmouth Algorithms Seminar, with Lorenz Wernisch, and with Andrzej Lingas.

†This study was supported in part by the funds of the National Science Foundation, CCR-9301714.

‡This study was supported in part by the funds of the National Science Foundation, DDM-9015851, and by a Fulbright Foundation grant that helps support this author's sabbatical visit to the Freie Universität.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

1.2 Background

Efficiently computing the greedy triangulation is a problem of long standing, going back at least to 1970 [9]. A number of the properties of the GT have been discovered [18, 22, 26, 27] and the greedy algorithm has been used in applications [5, 27].

A straightforward approach to computing the GT is to compute all $\binom{n}{2}$ distances, sort them, and then build the GT an edge at a time by examining each pair in order of length and adding or discarding it based on its compatibility with the edges already added. It is easy to see that this method requires $O(n^2)$ space and time

$$T(n) = O(n^2 \log n + n^2 f(n) + ng(n)) \qquad (1)$$

where $O(n^2 \log n)$ is the time required for an optimal comparison-based sort on $\binom{n}{2}$ distances, $f(n)$ is the time required to test new edges for compatibility, and $g(n)$ is the time required to update the data structure when a new greedy edge is added [29]. A naive test would compare each new potential edge to each of the existing edges (of which there are at most $O(n)$) for an $O(n^3)$ time algorithm. Gilbert [10] presented a data structure allowing an $O(\log n)$ time compatibility test and an $O(n \log n)$ time update, thus improving the algorithm's overall time complexity to $O(n^2 \log n)$, without adversely affecting space complexity. He does this by building a segment tree for each point in the set, where the endpoints of the "segments" are the polar angles between the given point and every other point in the set. Manacher and Zobrist [27] have since given an $O(n^2)$ expected time and $O(n)$ space greedy triangulation algorithm that makes use of a probabilistic method for *pretesting* compatibility of new edges. Note that our approach also uses this "generate and test" paradigm, and that we gain improvements over previous results by generating fewer edges and supplying more efficient tests.

An alternate approach to "generate and test" is to generate only compatible edges. One way to do this was discovered independently by Goldman [11] and by Lingas [24]. The method uses the generalized or constrained Delaunay triangulation [3, 15, 33]. The constrained Delaunay triangulation is required to include a set of edges E. The rest of the edges in the triangulation have the property that the circumcircle of the vertices of any triangle contains no point visible from all three vertices.

This alternate approach computes the constrained Deluanay triangulation of the points with the current set of GT edges as the set E. The next edge to be added to the GT can be found in linear time from the constrained Delaunay triangulation. The triangulation must then be updated to include the new edge in E, which takes $O(n \log n)$ time in the worst case. This gives an $O(n^2 \log n)$ time and $O(n)$ space algorithm, thus improving the space complexity of Gilbert's algorithm without affecting the worst case time. Lingas [24] shows that his method runs in $O(n \log^{1.5} n)$ for points chosen uniformly from the unit square.

Recently Levcopoulos and Lingas, and independently Wang, have shown how to do the update step in $O(n)$ time, using a modification of the linear-time algorithm for computing the Voronoi diagram of a convex polygon [1], leading to an $O(n^2)$ time and $O(n)$ space algorithm in the worst case [19, 31]. More recently Levcopoulos and Lingas give a modification of this algorithm that is expected to take $O(n)$ time for points uniformly distributed in a square [20]. These methods are elegant, but are significantly more complicated than our methods and should be slower for practical-sized problems.

One use of the greedy triangulation is as an approximation to the minimum weight triangulation (MWT). Given a set S of n points in the plane, a Minimum Weight Triangulation (MWT) of S is a triangulation that minimizes the total length of all edges in the triangulation. The MWT arises in numerical analysis [22, 25, 29]. In a method suggested by Yoeli [34] for numerical approximation of bivariate data, the MWT provides a good approximation of the sought-after function surface. Wang and Aggarwal use a minimum-weight triangulation in their algorithm to reconstruct surfaces from contours [32]. Though it has been shown how to compute the MWT in time $O(n^3)$ for the special case of n-vertex polygons [14], there are no known efficiently computable algorithms for the MWT in the general case [29]. We therefore seek efficiently computable *approximations* to the MWT.

Although neither the GT nor the Delaunay triangulation (DT) yields the MWT [26, 25], the GT appears to be the better of the two at approximating it. In fact, for convex polygons the GT approximates the MWT to with a constant factor while the DT can be a factor of $\Omega(n)$ larger [18]. For general point sets, the DT can be a factor of $\Omega(n)$ larger than the MWT, but the best lower bound for the GT is $\Omega(\sqrt{n})$ [13, 17]. For points lying on a convex polygon or uniformly distributed points in a square, both the GT and the DT are expected to be within a constant factor of the MWT [20, 4]. For these reasons a large amount of effort has gone into finding efficient methods for computing the greedy triangulation.

Other heuristics for approximating the MWT have also been developed. Plaisted and Hong have developed a complicated polynomial-time heuristic that is guaranteed to be within a factor of $O(\log n)$ of the MWT [28]. The best heuristics known so far are those of Lingas [23] and Heath and Pemmaraju [12]. These approaches make use of the convex hull and a spanning tree to create a single cell, and then use an optimal cell triangulation algorithm generalized from Gilbert's dynamic programming approach for computing the MWT of points on a convex polygon [10]. However, though both methods appear empirically to produce triangulations much better than the GT, both of these algorithms require $O(n^3)$ time which is impractical for large point sets. In [12], the authors only report on data for sets of size 50 and smaller. We also note that with the method of Heath and Pemmaraju, the greedy triangulation still remains of interest as it is a substep in their algorithm.

1.3 Notation

Throughout the paper, we let $d(p, q)$ be the distance from point p to q using the standard Euclidean distance metric.

2 A New Edge Test for the GT

We now present our new method for testing the compatibility of edges in a greedy triangulation. We will first present some definitions, and then a new theorem stating a property of any pair of points that does *not* form an edge in the GT. Following the theorem, we will present the edge test method with a proof of its correctness and a complexity analysis of the run time required by each of the operations.

Definition 1 *In a straight line planar graph T, a* clockwise chain *from p_1 (hereafter written "CW chain from p_1") is a sequence of points $p_1, \ldots, p_k$ such that for $1 \leq i < k, p_i p_{i+1}$ are edges in T, and for $1 < i < k, p_i p_{i+1}$ is the next edge around point p_i in a clockwise direction from $p_{i-1} p_i$.*

Definition 2 *Let $p_1, \ldots, p_k$ be a CW chain in straight line planar graph T. If (p_1, p_2) is the first edge in a clockwise direction from a segment $p_1 q$ (with $p_1 q$ not necessarily an edge in T), then we say that $p_1, \ldots, p_k$ is a* CW chain with respect to *$p_1 q$.*

We define a counter-clockwise chain, or CCW chain, in a similar fashion.

The new compatibility test method is based on the following lemma and theorem.

Lemma 1 *Given a set S of n points, let x, y, and z be oriented so that they form a CW triangle. Let T be the partial triangulation in the standard greedy algorithm at the time when xz is tested for compatibility or at some later time. Let x, y, z be a CCW chain in T. Then triangle xyz contains no points in S in its interior and xz is either compatible with T or already in T. (The lemma is also true if CW and CCW are interchanged.)*

Theorem 1 (CW/CCW Chain Theorem) *Given a set S of n points and a non-negative integer $k \leq \binom{n}{2}$, let T be a partial greedy triangulation of S constructed using the standard greedy approach of examining the first k interpoint pairs in nondecreasing order of distance and adding to T exactly those edges compatible with previously added edges. Now let (p, q) be the $(k+1)^{st}$ pair to be tested with $\delta = d(p, q)$. If edge pq is not compatible with T, then T contains a CW or CCW chain $p, v_1, \ldots, v_k, x$ (or, respectively, $q, v_1, \ldots, v_k, x$) with respect to pq that has the following five properties:*

1) edge $v_k x$ intersects segment pq,

2) points $v_1, \ldots, v_k$ lie within the $\delta \times 2\delta$ rectangle R that is divided into two squares by segment pq and $d(p, v_i) < \delta$ (respectively $d(q, v_i) < \delta$) for $1 \leq i \leq k$,

3) if $d(q, v_1) \geq \delta$ (respectively $d(p, v_1) \geq \delta$) then $k = 1$,

4) the vertices $p, v_1, \ldots, v_k$ (all edges of the chain before $v_k x$) form a convex chain, and

5) $v_k x$ is the closest edge to p (respectively q) of all edges in T intersecting pq.

2.1 Method and Proof of Correctness

We now give a fast greedy triangulation edge test based on Theorem 1. To determine whether edge pq is compatible, the algorithm examines the CW and CCW chains from p and q with respect to pq. The method actually requires that we examine at most only two edges on each of these chains. It returns TRUE if the edge is compatible and FALSE otherwise. For reasons of brevity, we give only the first part of this test: checking the CCW from p. Steps 2, 3, and 4 are symmetric except that in Step 4 instead of a goto Step 5 the algorithm would return TRUE. In part **1e.**, C_q is the circle of radius $d(p, q)$ centered at q.

2.1.1 Edge Test for (p, q) in T

```
Step 1: CCW Chain from $p$
    1a. Let $v_1, v_2$ be the next two points on
        the CCW chain from $p$ (with respect to $pq$).
    1b. IF ($\angle qpv_1 \geq \pi/2$) OR (no CCW chain exists)
        THEN goto Step 2.
    1c. IF $v_1 v_2$ intersects $pq$ THEN return FALSE.
    1d. IF $v_2 = q$ THEN return TRUE
    1e. IF $v_1$ is strictly inside $C_q$ AND $v_2 \neq q$
            THEN return FALSE.
            ELSE goto Step 2.
```

Proof of Correctness We now show that the above algorithm correctly determines whether edge pq is compatible with the edges already added to T (assuming the edges are added in greedy fashion.) We give a proof for Step 1; the other three steps are exactly analagous.

The first two cases are obvious. First assume that v_1 is outside R (or equivalently that no CCW chain exists). It follows directly from Theorem 1 that if pq is not compatible we will find a CW or CCW chain on one of the subsequent steps and we can immediately go to Step 2. Now assume that v_1v_2 intersects pq. By definition, pq is not compatible with T and we can halt.

Now assume that $v_2 = q$. That is, the second edge on the CCW chain is v_1q. In this case Lemma 1 says that pq is compatible, so we can halt and answer TRUE.

If v_1 is strictly inside C_q and $v_2 \neq q$, then we claim that pq can not be compatible with T. Though we have not *yet* found an existing edge v_kx intersecting pq, we know that one exists, because assuming otherwise would lead to a contradiction. Suppose that pq were compatible and were added to the triangulation. Then q, p, v_1 would be a CCW chain, so by Lemma 1 triangle qpv_1 would contain no points in its interior and v_1q would be compatible. Because it is shorter than pq it would already be in the triangulation. But then v_1q would be the next edge CCW around $v1$ from pv_1, contradicting the assumption that $v_2 \neq q$. Therefore pq cannot be compatible.

There is one remaining case, which is when v_1 is inside rectangle R but not strictly inside circle C, $v_2 \neq q$, and v_1v_2 does not intersect pq. That this CCW chain will not lead to *any* edge intersecting pq follows directly from Condition (3) of Theorem 1.

We have now completed the proof of the correctness of our greedy edge test method. □

2.2 Implementation and Analysis

We now discuss the implementation of our edge test and analyze its efficiency. We store our greedy triangulation T as a graph using adjacency lists. Each adjacency list is represented as a circular linked list of edges ordered in polar (or rotational) order around the point. To find CW and CCW chains from a new edge we must determine where in rotational order the new edge would fit. That would normally take $O(n)$ time, or $O(\log n)$ if we stored the circular list in a binary tree. Fortunately, we can use properties of the greedy triangulation to do better.

Let p be a vertex in T. We consider the neighbors of p (in T) in CCW order. Let x and y be two consecutive neighbors, and let α be the CCW angle swept from px to py. We call the ordered pair (x, y) *closed* if $\alpha < \pi$ and xy is in T, and we call (x, y) *open* otherwise. (Note that if (x, y) is closed, by Lemma 1 the triangle pxy must be empty, so no new greedy edges can connect to p between px and py.) We call an edge an *open edge* if it connects p to a point in an open ordered pair. A *closed interval* is a wedge around p that is bounded by two adjacent open edges whose endpoints in CCW order do not form an open pair. (If the edges around p are viewed as spokes of a wheel, then the open pairs will correspond to pairs of spokes with no "rim" between them and the closed intervals will be maximal sections of the wheel with the entire "rim" present.) A *closed point* is a point incident to at least one edge but to no open edges.

To the each edge structure in the data structure for storing the edges as a circular linked list we add pointers to the next CW and CCW open edge and a flag to show whether the wedge lying between that edge and the next open edge CCW from it is an open ordered pair or a closed interval. These new fields will only be maintained for open edges, and will provide a doubly-linked circular list of open edges around each point.

Maintaining this structure when a new edge pq is added to T is fairly straightforward. We will discuss updates at p; q is symmetric. If pq is the first edge incident to p, the new edge will point to itself as an open ordered pair spanning 2π. If pq is added to the middle of an open pair (x, y) (note that new edges can only be added between edges of an open ordered pair), we must do two types of updates. First, (x, y) must be split into two pairs (x, q) and (q, y), which may be open or closed. To find out which, we must see if edges xq and yq are in T, which can be done in constant time. Second, we must see if the new edge closes off a previously open pair. To do that, we see if the CCW edge from pq has the same endpoint w as the CW edge from qp (and the symmetric case on the other side). In either case, we must be able to update the data structure so that a previously open interval (x, y) is now closed. To do this the new closed interval (x, y) must be merged with possible closed intervals lying to either side of it to form a single closed interval. All of these tests and updates can be done in constant time.

With this data structure, the edge test is done as follows. The new edge pq is located in the circular list of open edges at p and q. If pq falls in a closed interval for either point, then the edge is not compatible. Otherwise perform the test described in the previous subsection, using the open edges as the first edges in the CW and CCW chains. This test will take time proportional to the number of open edges.

Fortunately, the maximum number of open edges adjacent to any point p is 10. This is because any pair (x, y) with $\alpha < \pi/3$ must be closed. Because α is not the largest angle in triangle pxy, we know that xy is not the longest edge in the triangle. By Lemma 1, xy is in T, which means that (x, y) is closed. Therefore every open pair has $\alpha \geq \pi/3$, so so there cannot be more than six of them around p. If there are six, each edge is shared by two open intervals, so there are only six edges. If there are five or fewer open pairs, then there cannot be more than 10 open edges.

To initialize the data structure we simply create n empty circular lists in $O(n)$ time. Because there are $O(n)$ edges in the entire triangulation the total space required by all lists is $O(n)$.

Thus we have a data structure that requires $O(1)$ time for an edge test or an update, $O(n)$ time to initialize, and $O(n)$ space. For comparison, we end this subsection by noting that the method of Gilbert [10] requires $O(n^2 \log n)$ time for initialization, $O(n \log n)$ time for update, and $O(n^2)$ space.

3 A Necessary Condition for an Edge to be in the Greedy Triangulation

We begin by stating a simple and obvious lemma.

Lemma 2 (Convex Quadrilateral Rule) *Let pqr and pqs be two empty triangles in a greedy triangulation. If segment rs intersects segment pq (that is, prqs is a convex quadrilateral), then* $d(p,q) \leq d(r,s)$.

Heath and Pemmaraju [12] describe this as the property of "local optimality."

We now give an important (though less obvious) lemma[1] that states a necessary (but not sufficient) condition for an edge to be a greedy triangulation edge. This is a variant of Lemma 3.1 in [21].

Lemma 3 *Let p, q be a pair of points in a set S. Consider the disc D of radius $\delta = d(p,q)/(2\sqrt{5})$ centered at the midpoint of pq. Let pq divide the disc into two half-disks. If both half-disks contain at least one point in S then pq cannot be in the GT of S.*

Proof Sketch For notational convenience, orient the plane so that segment pq is horizontal, with p on the left side. (For the remainder of the proof, refer to Figure 1.) Let a be the point closest to pq in the upper half-disk of D, and let b be the point closest to pq in the lower half-disk of D. Let C be the circle with pq as a diameter, and let α be the length of the shortest segment with endpoints on or outside of C that passes through D. Then $\alpha = \sqrt{d(p,q)^2 - 4\delta^2}$.

We will give a proof by contradiction making use of the following observation.

Observation 1 Let p_1 and p_2 be any pair of points that are not connected by an edge in the GT. Then some GT edge intersecting segment p_1p_2 must have length $\leq d(p_1, p_2)$.

Assume pq is a GT edge. Then ab cannot be a GT edge, and by Observation 1 there must be some GT edge of length less than $d(a,b) \leq 2\delta$ "cutting off" ab. We will show that no such edge exists either above or below pq.

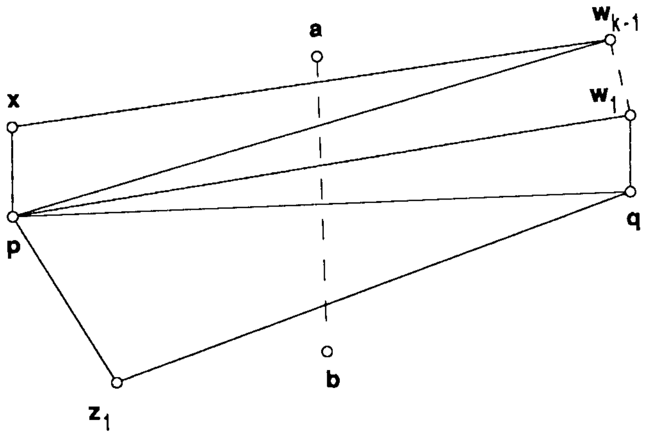


Figure 1: Edges intersecting ab

Since there are points on both sides of edge pq, it is not a CH edge, and therefore it must be an edge in two triangles – an upper and a lower. Let pw_1q be the upper triangle and let pz_1q be the lower triangle. If w_1 and z_1 are both in C, then $d(w_1, z_1) < d(p,q)$ and thus by Lemma 2 pq could not be a GT edge and we have a contradiction. Without loss of generality, we therefore assume that w_1 is outside of C and is closer to q than to p. Since edge pw_1 cuts off ab, it follows that $d(p, w_1) \geq \alpha$.

For notational convenience, we also label all the GT edges intersecting segment ab. The GT edges above pq that intersect ab we will label $e_1, \ldots, e_m$ and the edges below ab that intersect ab we label $f_1, \ldots, f_n$. We now give a case analysis on these edges. First, all of these edges cannot have both endpoints outside of or on C, because they comprise all of the edges crossing ab, and all of them would be longer than α. But our choice of δ causes α to be greater than 2δ, and by Observation 1, one intersecting GT edge must be shorter than $d(a,b)$.

The final two observations lead to our contradiction.

Observation 2 At least one e_i and at least one f_j must have an endpoint in C.

Observation 3 If e_k is the first e_i to have an endpoint in C, then $e_1, \ldots, e_{k-1}$ are all of the form pw_i for $1 \leq i \leq k-1$, and e_k is of the form xw_{k-1}. Furthermore, x lies to the left of D. Finally, for some $1 \leq l \leq k-1$ there is an point w_l that lies within $d(p,q)$ of p. (See Figure 1 again.)

[1] an earlier false version of this lemma with $\delta = d(p,q)/2$ was given in [6]. We give the corrected version here.

The Contradiction This structure is very constraining. We have constrained x to lie to the left of D and inside of C. Furthermore, if x were closer to q than α, then by Observation 1 one of the e_i with $1 \leq i \leq k-1$ would have to be shorter than α, which cannot be the case. (None of them have endpoints in C.) Therefore x is at least α from q. Finally, x must lie below the ray from q that is tangent to D from above.

We have also constrained point w_l to lie outside of C near q, but inside the circle of radius $d(p,q)$ centered at p. It must also lie below the ray from from p tangent to D from above.

Observation 2 says that some edge f_j will be the first edge below pq along ab with an endpoint y inside of C. But where can y lie? It must lie above either the ray from p tangent to D from below or the ray from q tangent to D from below. If it is in the left half of C, its distance to x will be less than α. If it is in the right half of C, its distance to w_l will be less than α. In either case we have a contradiction, because all intervening edges will be at least α long. (All of them have both endpoints on or outside of C.) □

3.1 Comments on the Constant

The proof above requires α to be larger than a number of different values. The strongest requirement arises in the last case, when we let y lie at the intersection of the ray from p tangent to D from below and C and we let w_l lie at the intersection of the ray from p tangent to D from above and the circle of radius $d(p,q)$ centered at p. The actual bound of $d(p,q)/(2\sqrt{5})$ is derived from a slightly worse configuration, which cannot be actually achieved. We require that the intersection points between the two rays from p tangent to D and the circle of radius $d(p,q)$ centered at p be at most α apart. This case is less than 5 percent worse than the actual worst case, and is less complicated to calculate and express.

We do not believe that our bound is tight. The worst example that we have been able to find is a diamond with pq as its diameter, with two additional points just outside of the midpoints of two opposite edges of the diamond. This case approaches the bound $\delta = d(p,q)/(2\sqrt{2})$ as closely as is desired.

4 An Analysis of the Edges in the Greedy Triangulation

For two points p and q in the plane, and for a real number $r > 0$, let $D(p,q,r)$ denote the closed disk of radius r centered at $(p+q)/2$. The line through p and q defines two closed semidisks of $D(p,q,r)$, denoted by $D'(p,q,r)$ and $D''(p,q,r)$ (without specifying which half is D' and D'', respectively).

Let us employ a constant $0 < \gamma \leq 1$, fixed for the whole section. Given a set S of n points in the plane, we call a pair $\{p,q\}$ of points in S ***plausible***, if $D'(p,q,r_1) \cap S = \emptyset$ or $D''(p,q,r_1) \cap S = \emptyset$, with $r_1 = \gamma|p-q|/2$. The previous section showed that if $\gamma = 1/\sqrt{5}$, only plausible edges can be in a greedy triangulation. When $\gamma = 1$ only plausible edges can be Delaunay, so the lemmas we prove about distributions of edge lengths apply to both greedy and Delaunay triangulations.

Our first goal is to estimate the expected number of plausible pairs in a set S of n points, uniformily distributed in a convex compact region $\mathcal{C}$. We normalize $\mathcal{C}$ to be of area 1 to simplify the notation.

Suppose p and q are points in $\mathcal{C}$, so that $D(p,q,r_1)$ is contained in $\mathcal{C}$ (r_1 as above). If we choose another $n-2$ points at random from $\mathcal{C}$, then the probability of $\{p,q\}$ being plausible is the same as the probability that $D'(p,q,r_1)$ is empty or $D''(p,q,r_1)$ is empty. This gives us the following:

$$\begin{aligned} & Prob(\{p,q\} is\ plausible) \\ & \quad \leq \quad Prob(D'\ is\ empty) + Prob(D''\ is\ empty) \\ & \quad = \quad 2\left(1 - \frac{r_1^2\pi}{2}\right)^{n-2} \\ & \quad < \quad 2e^{-r_1^2\pi(n-2)/2} . \end{aligned}$$

We have to cope with pairs of points $\{p,q\}$ where $D(p,q,r_1)$ extends beyond the boundaries of $\mathcal{C}$: For $p \in \mathcal{C}$, let $\beta(p)$ denote the distance of p from the boundary of $\mathcal{C}$, or, in other words, the largest radius of a disk centered at p which is still contained in $\mathcal{C}$. Then we observe that for any pair if points p and q in $\mathcal{C}$, the disk $D(p,q,r_2)$, $r_2 = (\beta(p)+\beta(q))/2$, is contained in $\mathcal{C}$. It follows that for any pair $\{p,q\} \subset \mathcal{C}$ the probability of being plausible (after adding (n-2) random points) is bounded by

$$2e^{-r^2\pi(n-2)/2},$$

for $r = r_{p,q} := \min\{r_1, r_2\}$, with r_1 and r_2 dependent on p and q as previously specified. Now let p and q be two random points in a random set of n points in $\mathcal{C}$. Then the probability of being plausible is bounded by

$$\begin{aligned} & \int_{p\in\mathcal{C}} \int_{q\in\mathcal{C}} 2e^{-r^2\pi(n-2)/2}\, dq dp \\ & \quad < \int_{p\in\mathcal{C}} \int_{q\in\mathcal{C}} 2e^{-r_1^2\pi(n-2)/2}\, dq dp \\ & \qquad + \int_{p\in\mathcal{C}} \int_{q\in\mathcal{C}} 2e^{-r_2^2\pi(n-2)/2}\, dq dp . \qquad (2) \end{aligned}$$

In order to estimate the terms in (2) we use density functions $f_p(x)$, $p \in \mathcal{C}$, and $g(x)$. $\int_{-\infty}^{z} f_p(x)dx$ is the probability for a random point in $\mathcal{C}$ to have distance at most z from p; clearly, $f_p(x) \leq 2x\pi$ for all $p \in \mathcal{C}$ and $x \geq 0$. $\int_{-\infty}^{z} g(x)dx$ is the probability of a random point in $\mathcal{C}$ to have distance at most z from the boundary of $\mathcal{C}$;

here we have a bound of $g(x) \leq U$, U is the perimeter of $\mathcal{C}$, for all $x \geq 0$.

Now the left term in (2) equals

$$\begin{aligned} & \int_{p\in\mathcal{C}} \int_{x=0}^{\infty} 2f_p(x)e^{-x^2\gamma^2\pi(n-2)/8}\,dxdp \\ \leq & \int_{x=0}^{\infty} 4x\pi e^{-x^2\gamma^2\pi(n-2)/8}\,dx \\ = & \frac{16}{\gamma^2(n-2)} \;. \end{aligned}$$

The right term in (2) equals

$$\begin{aligned} & \int_{x=0}^{\infty} g(x) \int_{y=0}^{\infty} g(y)2e^{-(x+y)^2\pi(n-2)/8}\,dydx \\ \leq & \;2U^2 \int_{x=0}^{\infty}\int_{y=0}^{\infty} e^{-(x+y)^2\pi(n-2)/8}\,dydx \\ = & \frac{8U^2}{\pi(n-2)} \;. \end{aligned}$$

Lemma 4 *Let X be the random variable for the number of plausible pairs in a random set S of $n > 4$ points uniformly distributed in a convex region of area 1 and perimeter U. Then*

$$E(X) < 4(n+2)(\frac{U^2}{\pi} + \frac{2}{\gamma^2}) = O(U^2 n)$$

and

$$E(X \log X) \leq 2E(X) \log n = O(U^2 n \log n) \;.$$

Let us call a pair $\{p,q\} \subset \mathcal{C}$ a *candidate* (for being plausible), if $r = r_{p,q} \leq B$, where $B := \sqrt{(c \ln n)/(n-2)}$ for a constant c. Our next goal is now to show, that a random set does not contain too many candidates, and–with high probability–all plausible pairs are indeed candidates. We use a similar method of splitting the problem into inequalities, and arrive at the following lemma:

Lemma 5 *Let Y be the random variable for the number of candidates in a random set S of $n > 4$ points uniformly distributed in a convex region of area 1 and perimeter U. Then*

$$E(Y) < \left(\frac{2\pi}{\gamma^2} + U^2\right) c(n+2)\ln n = O(U^2 cn \log n) \;.$$

The probability that there exists a plausible pair that is not a candidate is bounded by

$$n^{2-c\pi/2} \;.$$

5 Greedy Triangulation Algorithms

These results lead to the following greedy triangulation algorithm:

5.1 Algorithm 1

Step 1: Generate all plausible pairs with $r_1 \leq B$. To do this, we generate all pairs of points separated by a distance of at most $2B/\gamma$. This is the fixed-radius-near-neighbors problem [2, 7]. In this case a bucketing algorithm by Bentley, Stanat, and Williams can solve the problem in time $O(n+m)$, where m is the number of pairs that lie within $2B/\gamma$ of one another. As each pair is generated, test to see if it is plausible using the method described below.

Step 2: Generate all plausible pairs with $r_2 \leq B$. The easy way to do this is to find all points within $2B$ of the boundary of $\mathcal{C}$ and to generate all pairs. However, this can generate pairs with r_2 as long as $2B$. By sorting the points within $2B$ of the boundary of $\mathcal{C}$ in order of their distance from the boundary one can generate only the pairs needed by matching each point with points further from the boundary than itself only until $r_2 > B$, then going on to the next point. Also, note that pairs that also have $r_1 \leq B$ can be ignored, because they were generated in Step 1. Test each pair to see if it is plausible.

Step 3: Generate the greedy triangulation of the plausible pairs generated in Steps 1 and 2. To do this, first sort the pairs in increasing order of distance between the points. Start with an empty triangulation, and attempt to create an edge between each pair in turn, failing to create the edge if it fails the compatibility test described above.

5.2 Testing to See if a Pair is Plausible

We need a fast test to see if an edge is plausible. One way to do this uses a grid of squares. As a preprocessing step, cover $\mathcal{C}$ by a grid of $O(n)$ squares, each with side $1/\sqrt{n}$. For each bucket, create a list of the points in S that fall in that bucket. Then to test a pair (p,q), compute $D(p,q,r)$, with $r = \min(r_1, r_2)$ as described above. Go through the squares that overlap $D'(p,q,r)$ until you find a point lying in $D'(p,q,r)$ or find that no such point exists. Similarly go through the squares that overlap $D''(p,q,r)$ until you find a point lying in this half-disk or find that no such point exists. If either half-disk is empty, the point is plausible. In searching through the grid squares, start with the squares with maximum overlap with the half-disk.

For uniformly distributed points this test will run in $O(1)$ expected time. The probability that a grid square that lies completely in $\mathcal{C}$ contains no point from S is $(1 - 1/n)^n$. This is always less than e^{-1}. Therefore the probability that a given grid square contains a point is greater than $1 - e^{-1} > 0.63$. This implies that a search through grid squares looking for a non-empty one is expected to look at fewer than 2 squares. When r is small enough that no full grid squares overlap a half-disk, then only a constant number of grid squares overlap the half-disk. When r is larger, the number of grid squares examined before an point is found is expected be a constant.

Unfortunately, a bad distribution of points could cause this test to look at $O(n)$ squares for a very long edge. To prevent this, we stop testing after looking at $c \ln n$ squares for some c that we choose. Thus the test could occasionally decide that a half-disk is empty when it in fact is not. Thus a candidate could occasionally be considered plausible when it is not. But this will happen with probability less than $(1-e^{-1})^{c \ln n} = n^{c(\ln(e-1)-1)}$. This is about $n^{-.46c}$. We can therefore choose c so that the expected number of edges on which the test fails is $o(1)$, and this limitation will avoid bad worst-case times without hurting expected times.

5.3 Analysis of Algorithm 1

Given this test for plausible pairs, it is easy to show that Algorithm 1 runs in $O(n \log n)$ expected time. By using the floor function we can do the preprocessing for the edge test in $O(n)$ time. By Lemma 5 the number of pairs considered in Steps 1 and 2 is $O(n \log n)$. Each test for plausibility takes constant expected time, so these steps require $O(n \log n)$ time. By Lemma 4 the number of pairs that are generated in Steps 1 and 2 that are plausible is $O(n)$. Sorting these takes $O(n \log n)$ time, and the compatibility test for each takes $O(1)$ time. Therefore Step 3 and the entire algorithm require $O(n \log n)$ time and $O(n)$ space. In the worst case the algorithm could take $O(n \log^2 n)$ time and $O(n \log n)$ space.

Unfortunately, this algorithm is not guaranteed to generate the greedy triangulation. It will generate it with probability $1-n^{2-c\pi/2}$, and because we can choose c as large as we like we can make this probability arbitrarily close to 1. But the algorithm could fail to produce a triangulation, either because neither diagonal of a quadrilateral was a candidate or because the edge compatibility test (which depends on the correctness of the partial triangulation) could fail if edges were missing. Alternately, it could produce a triangulation that was not the greedy one, and there is no known fast way to verify the correctness of a greedy triangulation.

Fortunately, a minor modification of the algorithm will eliminate this problem.

5.4 Algorithm 2

We turn Algorithm 1 into a two-phase algorithm. We run Steps 1 and 3 of Algorithm 1, but skip Step 2. What we will be left with at the end of this process is a partial greedy triangulation. All of the short edges will be present, but no edge longer than $2B/\gamma$ will be present. But Lemma 5 implies that with very high probability, all edges that have either endpoint at least $2B/\gamma$ from the boundary of $\mathcal{C}$ will be present. We will call points at least $2B/\gamma$ from the boundary *interior points.* We call a greedy triangulation edge with an interior point as (at least) one of its endpoints an *interior edge.* The expected number of interior edges that are missing is bounded by the expected number of plausible edges which are not candidates, so is at most $n^{2-c\pi/2}$. As long as c is chosen to be at least $4/\pi$ this is $O(1)$ edges. This implies that at most $O(1)$ interior points will not be closed (as defined in the edge test section). This is because a missing edge can cause at most four points to be not closed (the four vertices of the quadrilateral with the missing edge as diagonal).

This observation leads to a new Step 2′.

Step 2′: Generate plausible long pairs Generate all possible pairs of non-closed points, and reject all implausible pairs. Sort these pairs, and continue running Step 3 of Algorithm 1 with these pairs as well.

Because all short pairs are tried and then all longer pairs that could possibly create an edge are tried, the algorithm will correctly generate the greedy triangulation of any set of points.

The edge test data structure keeps track of the list of incident open edges for each point. A point which is closed will have incident edges, but no open edges in its list.

Will this step generate too many candidate pairs? The number of interior points that are not closed is $O(1)$. Non-interior points lie within $2B/\gamma$ of the boundary of $\mathcal{C}$. The total number of non-interior points is expected to be at most $2BUn/\gamma$, because $2BU/\gamma$ is an upper bound on the area close enough to the boundary of $\mathcal{C}$. Therefore the total number of candidate edges generated in Step 2′ is $\binom{2BUn/\gamma+O(1)}{2} = O(n \log n)$.

Lemma 4 says that the total expected number of plausible pairs is $O(n)$, so Step 2′ is expected to generate $O(n)$ plausible pairs. Therefore sorting and testing these for compatability will take $O(n \log n)$ time.

This shows that Algorithm 2 will always compute the greedy triangulation and is expected to run in $O(n \log n)$ time, using $O(n)$ space. In the worst case it could take $O(n^2 \log n)$ time and $O(n^2)$ space.

5.5 Algorithm 3 – An Algorithm that Depends Less on the Uniform Distribution

Algorithm 2 depends heavily on the uniform distribution, both to get a fast expected-time test for plausibility and to tell when to end the first phase and begin the second. The following algorithm is a variant that is less sensitive to the exact distribution. It will dynamically decide when to switch from one phase to the next in an attempt to balance the amount of work done in each phase.

In the first phase it generates possible edges in increasing order using an algorithm of Dickerson, Drysdale, and Sack [8]. (Algorithms to enumerate the k closest interpoint pairs have been invented by by Salowe and by Lenhof and Smid, but because they need to know k in advance they are less appropriate in this context [30, 16].) When the number of pairs of not closed points is proportional to the number of pairs already examined, it starts over, enumerating pairs of not closed points in increasing order (similar to Step $2'$ above).

How do we know when to switch over? We keep track of the number of points which are not closed. When during an edge insertion a point becomes closed, we subtract 1 from the number of non-closed points c. When $c^2/2$ is greater than the number of edges tested so far, the number of edges generated in the second phase will equal the number in the first, so we change to the second phase.

5.6 Analysis of Algorithm 3

Generating the next pair in increasing order requires $O(\log n)$ time, and a compatibility test takes $O(1)$ time. Therefore the algorithm runs in time $O(\log n)$ times the number of pairs generated. It requires space proportional to the number of pairs generated.

For the uniform distribution, we have already seen that all interior points are expected to be closed after examining $O(n \log n)$ edges, and that at that point the number of points which are not closed is $O(Bn) = O(\sqrt{n \log n})$. Therefore the number of pairs in the second phase will be $O(n \log n)$. This implies that the algorithm is expected to take $O(n \log^2 n)$ time and $O(n \log n)$ space for a uniform distribution.

6 Summary and Open Problems

We have given a new method for testing edge compatibility in a greedy triangulation. The method is based on Theorem 1, the CW/CCW chain theorem, which states an interesting property of greedy triangulations. Our method requires only $O(1)$ time for both the compatibility test and updates operations. This is a significant improvement over previous methods.

We then proved a necessary condition involving half-disks for an edge to be in the greedy triangulation. This lead to theorems on the number of pairs of points that were plausible and that were candidates to be plausible edges.

Finally, we used these characterizations and the compatibility test to prove the correctness and runtime of several new algorithms for computing the greedy triangulation. On uniformly distributed points we can compute the greedy triangulation in expected time $O(n \log n)$ and space $O(n)$.

Some obvious questions arise from this work.

Problem 1 *OPEN We can construct a point set for which Algorithm 3 would require $\Theta(n^2 \log n)$ time. However this set is highly structured and non-random. What is the expected run time for Algorithm 3 for random distributions other than the uniform distribution?*

Problem 2 *OPEN Is there a greedy triangulation algorithm requiring $o(n^2)$ time in the* worst *case? Note that with our edge test, a way to enumerate all pairs of points in increasing order by distance in $O(n^2)$ time would lead to an $O(n^2)$ algorithm for the GT, but would not lead to an $o(n^2)$ algorithm.*

Problem 3 *OPEN What is the true worst-case ratio for δ in Lemma 3? We have bounded it between $d(p,q)/(2\sqrt{5})$ and $d(p,q)/(2\sqrt{2})$.*

References

[1] A. Aggarwal and L. J. Guibas and J. Saxe and P. Shor, "A linear time algorithm for computing the Voronoi diagram of a convex polygon ." *Discrete Comput. Geom.* **4** (1989) 591–604.

[2] J. Bentley, D. Stanat and E. Williams Jr., "The complexity of finding fixed-radius near neighbors." *Information Processing Letters* **6** (1977) 209–213.

[3] P. L. Chew, "Constrained Delaunay triangulations." *Proceedings of the Third Annual ACM Symposium on Computational Geometry* (1987) 215–222.

[4] R. C. Chang and R. C. T. Lee, "On the average length of Delaunay triangulations." *BIT* **24** (1984) 269–273.

[5] F. Dehne, Flach, J.-R. Sack, and R.Valiveti, "Analog Parallel Computational Geometry." *Proceedings of the 5th Canadian Conference on Computational Geometry* (1993) 143–153.

[6] M. Dickerson, "Expected rank of the longest edge in the greedy triangulation." *Proc. of the 4th Canadian Conference on Computational Geometry* (1992) 182–187.

[7] M. Dickerson and R. S. Drysdale, "Fixed-radius near neighbors search algorithms for points and segments." *Information Processing Letters* **35** (1990) 269–273.

[8] M. Dickerson, R.L. Drysdale, and J-R Sack, "Simple Algorithms for enumerating interpoint distances and finding k nearest neighbors." *International Journal of Computational Geometry and Applications* **3** (1992) 221–239.

[9] R. D. Düppe and H. J. Gottschalk, "Automatische Interpolation von Isolinien bei willküerlich verteilten Stützpunkten." *Allgemeine Vermessungsnachrichten* **77** (1970) 423–426.

[10] P. Gilbert, "New results in planar triangulations." MS Thesis, University of Illinois, Urbana, IL, 1979.

[11] S. Goldman, "A Space Efficient Greedy Triangulation Algorithm." *Information Processing Letters* **31** (1989) 191–196.

[12] L. Heath and S. Pemmaraju, "New results for the minimum weight triangulation problem." *Virginia Polytechnic Institute and State University, Department of Computer Science, TR 92-30* (1992). To appear in *Algorithmica*.

[13] D. Kirkpatrick, "A note on Delaunay and optimal triangulations." *Information Processing Letters* **10** (1980) 127–128.

[14] G. Klincsek, "Minimal triangulations of polygonal domains." *Ann. Discrete Math.* **9** 121–123.

[15] D. T. Lee and A. K. Lin, "Generalized Delaunay triangulations for planar graphs." *Discrete Comput. Geom.* **1** (1986) 201–217.

[16] H. P. Lenhof, M. Smid, "Enumerating the k-closest pairs optimally." *Proceedings of the 33rd FOCS*, (1992) 380–386.

[17] C. Levcopoulos, "An $\Omega(\sqrt{n})$ Lower Bound for the Nonoptimality of the Greedy Triangulation." *Information Processing Letters* **25** (1987) 247–251.

[18] C. Levcopoulos and A. Lingas, "On Approximating Behavior of the Greedy Triangulation for Convex Polygons." *Algorithmica* **2** (1987) 175–193.

[19] C. Levcopoulos and A. Lingas, "Fast Algorithms for Greedy Triangulation." *BIT* **32** (1992) 280–296.

[20] C. Levcopoulos and A. Lingas, "Greedy Triangulation Approximates the Minimum Weight Triangulation and Can be Computed in Linear Time in the Average Case." Tech. Report LU-CS-TR:92-105, Dept. of Computer Science, Lund University, 1992.

[21] A. Lingas, "The greedy and Delaunay triangulations are not bad in the average case." *Information Processing Letters* **22** (1986) 25–31.

[22] A. Lingas, "On Approximating Behavior and Implementation of the Greedy Triangulation for Convex Planar Point Sets." *Proceedings of the Second Annual ACM Symposium on Computational Geometry* (1986) 72–79.

[23] A. Lingas, "A new heuristic for the minimum weight triangulation." *SIAM Journal of Algebraic and Discrete Methods* **8** (1987) 646–658.

[24] A. Lingas, "Greedy triangulation can be efficiently implemented in the average case." *Proceedings of Graph-Theoretic Concepts in Computer Science* (1988) 253–261.

[25] E. Lloyd, "On triangulations of a set of points in the plane." *Proceedings of the 18th FOCS* (1977) 228–240.

[26] G. Manacher and A. Zobrist, "Neither the greedy nor the Delaunay triangulation of the planar set approximates the optimal triangulation." *IPL* **9** (1979) 31–34.

[27] G. Manacher and A. Zobrist, "Probabilistic methods with heaps for fast-average-case greedy algorithms." *Advances in Computing Research* vol. 1 (1983) 261–278.

[28] D. A. Plaisted and J. Hong, "A heuristic triangulation algorithm." *J. Algorithms* **8** (1987) 405–437.

[29] F. Preparata and M. Shamos, "Computational Geometry: an introduction." Springer-Verlag, 1985.

[30] J.S. Salowe, "Enumerating distances in space." *Internat. J. Comput. Geometry Appl.* **2** (1992) 49–59

[31] C. A. Wang, "Efficiently updating the constrained Delaunay triangulations." *BIT* **33** (1993) 238–252.

[32] Y.F. Wang and J.K. Aggarwal, "Surface reconstruction and representation of 3-D scenes." *Pattern Recognition* **19** (1986) 197–207.

[33] C. A. Wang and L. Schubert, "An optimal algorithm for constructing the Delaunay triangulation of a set of line segment." *Proceedings of the Third Annual ACM Symposium on Computational Geometry* (1987) 223–232.

[34] P. Yoeli, "Compilation of data for computer-assisted relief cartography." In *Display and Analysis of Spatial Data* J.C.Davis and M.J. McCullagh, editors, John Wiley & Sons, NY (1975).

Linear-size Nonobtuse Triangulation of Polygons

Marshall Bern [*] Scott Mitchell [†] Jim Ruppert [‡]

Abstract

We give an algorithm for triangulating n-vertex polygonal regions (with holes) so that no angle in the final triangulation measures more than $\pi/2$. The number of triangles in the triangulation is only $O(n)$, improving a previous bound of $O(n^2)$, and the worst-case running time is $O(n \log^2 n)$. The basic technique used in the algorithm, recursive subdivision by disks, is new and may have wider application in mesh generation. We also report on an implementation of our algorithm.

1. Introduction

The triangulation of a two-dimensional polygonal region is a fundamental problem arising in computer graphics, physical simulation, and geographical information systems. Most applications demand not just any triangulation, but rather one with triangles satisfying certain shape and size criteria [9]. In order to satisfy these criteria, one typically allows triangles to use new vertices, called *Steiner points*, that are not vertices of the input polygon. The number of Steiner points should not be excessive, however, as this would increase the running time of computations.

Throughout the application areas named above, it is generally true that large angles (that is, angles close to π) are undesirable. Babuška and Aziz [2] justified this aversion for one important application area by proving convergence of the finite element method [25] as triangle sizes diminish, so long as the maximum angle is bounded away from π. They also gave an example in which convergence fails when angles grow arbitrarily flat. (An elementary example in which large angles spoil convergence is Schwarz's paradox [22].)

Any bound smaller than π implies convergence in Babuška and Aziz's model, but a bound of $\pi/2$ on the largest angle has special importance. First of all, any stricter non-varying requirement would also bound the smallest angle away from zero; for some inputs (such as a long, skinny rectangle) this forces the triangulation to contain a number of triangles dependent on the geometry—not just on the combinatorial complexity—of the input. Second, a nonobtuse triangulation is necessarily a (constrained) Delaunay triangulation [8]. Third, a nonobtuse triangulation admits a *perpendicular planar dual*, that is, an embedding in which dual edges cross at right angles. Such an embedding is convenient for the "finite volume" method [25]. Finally, a nonobtuse triangulation has better numerical properties [3, 27]. In particular, Vavasis [27] recently proved that for finite element problems with physical characteristics that vary enormously over the domain, a nonobtuse mesh implies faster convergence of a certain numerical method.

These properties have established nonobtuse triangulation as a desirable goal in finite element mesh generation. Several heuristic methods have been developed to compute nonobtuse triangulations [4, 20]. Baker, Grosse, and Rafferty [3] gave the first provably-correct algorithm. Their algorithm also bounds the smallest angle away from zero, and hence necessarily uses a number of triangles dependent upon input geometry. Melissaratos and Souvaine [16] gave another algorithm of this type.

From the point of view of theoretical computer science, however, it is important to determine the inherent complexity of nonobtuse triangulation, apart from no-small-angle triangulation. Bern and Eppstein [8] devised a nonobtuse triangulation algorithm using $O(n^2)$ triangles, where n is the number of vertices of the input domain. This result demonstrates a fundamental com-

[*] Xerox Palo Alto Research Center, 3333 Coyote Hill Rd., Palo Alto, CA 94304

[†] Applied and Numerical Mathematics Dept., Sandia National Laboratories, Albuquerque, NM 87185. This work was supported by the Applied Mathematical Sciences program, U.S. Department of Energy Research and by the U.S. Department of Energy under contract DE-AC04-76DP00789.

[‡] NASA Ames Research Center, M/S T27A-1, Moffett Field, CA 94035. The author is an employee of Computer Sciences Corporation, supported under NASA contract NAS 2-12961.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

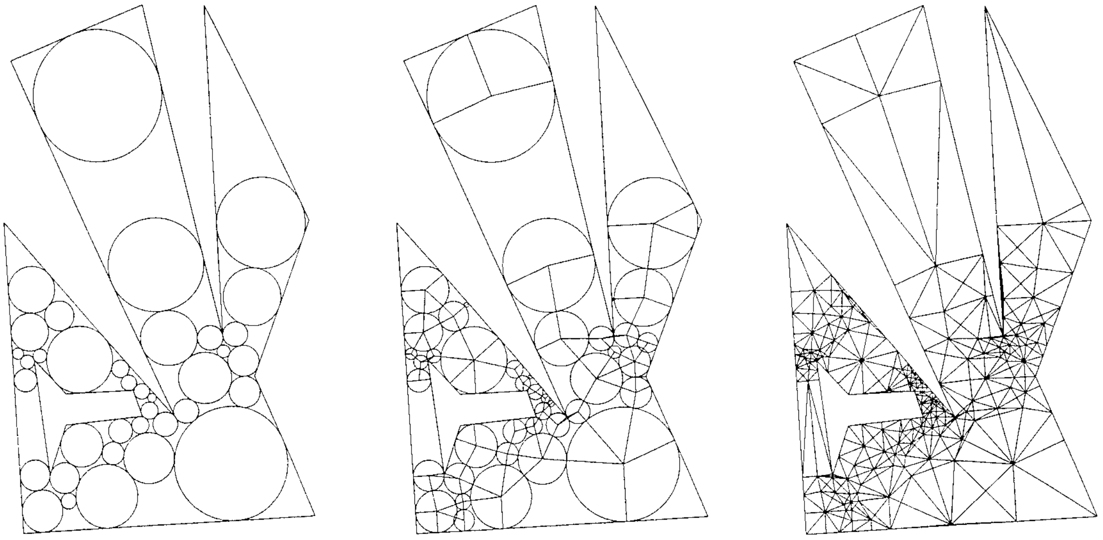


Figure 1. (a) Disk packing. (b) Induced small polygons. (c) Final triangulation.

plexity separation between bounding large angles and bounding small angles. Bern, Dobkin, and Eppstein [7] later improved this bound to $O(n^{1.85})$ for convex polygons.

In this paper, we improve these bounds to linear, using an entirely different—and more widely applicable—technique. Aside from sharpening the theory, our new algorithm boasts other advantages: it parallelizes, thereby placing nonobtuse triangulation in the class $\mathcal{NC}$; and it does not use axis-parallel grids, so the output has no preferred directions. Our algorithm also improves results of Bern et al. [7] on no-large-angle triangulation. The superseded results include an algorithm guaranteeing a maximum angle of at most $5\pi/6$ that uses $O(n \log n)$ triangles for simple polygons and $O(n^{3/2})$ triangles for polygons with holes.

2. Overview of the Algorithm

Our algorithm consists of two stages. The first stage (Section 3) packs the domain with non-overlapping disks, tangent to each other and to sides of the domain. The disk packing is such that each region not covered has at most four sides (either straight sides or arcs), as shown in Figure 1(a). The algorithm then adds edges (radii) between centers of disks and points of tangency on their boundaries, thereby dividing the domain into small polygons as shown in Figure 1(b).

The second stage (Section 4) triangulates the small polygons using Steiner points located only interior to the polygons or on the domain boundary. Restricting the location of Steiner points ensures that triangulated small polygons fit together so that neighboring triangles share entire sides. Figure 1(c) shows the resulting nonobtuse triangulation.

This algorithm is circle-based, rather than grid-based like the previous polynomial-size nonobtuse triangulation algorithm [8]. Analogously, the problem of no-small-angle triangulation has grid-based [10] and circle-based [23] solutions. In retrospect, circle-based algorithms offer a more natural way to bound angles, as well as meshes more intrinsic to the input domain. This conclusion is supported by two more examples. In the second stage of this paper's algorithm, certain misshapen small polygons cause technical difficulties; these are neatly solved by packing in more disks. (One of these additional disks is the second from the left along the bottom side of Figure 1(b).) In other recent work, Mitchell uses the "angle buffering" property of circles to give a triangulation, restricted to use only interior Steiner points, with linear size and largest angle nearly as small as possible [18].

3. Disk Packing

In this section, we describe the first stage of the algorithm. Let P denote the input: a region of the plane bounded by a set of disjoint simple polygons with a total of n vertices. An *arc-gon* is a simple polygon with sides that are arcs of circles. The circles may have various radii, including infinity (which implies a straight side).

Throughout the disk-packing stage, we make use of the *generalized Voronoi diagram* (GVD), which is defined by proximity to both edges and vertices. The interior points of polygonal region P are divided into cells

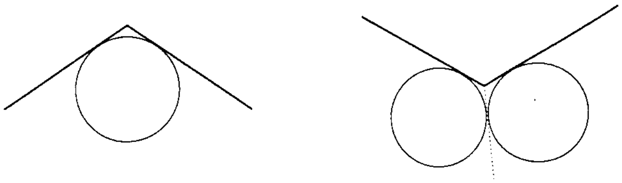


Figure 2. Adding disks at (a) convex and (b) concave corners of polygonal region P.

according to the nearest vertex of P, or the nearest edge (viewing each edge as an open segment). The resulting partition consists of a set of bisectors, either line segments or parabolic arcs; it is essentially the same as the *medial axis* [21]. The GVD can be similarly defined for arc-gons, or more generally for arbitrary collections of points, segments, and circular arcs. The GVD of a collection of n points, segments, and arcs can be computed in time $O(n \log n)$ [13].

The disk-packing stage consists of three smaller steps. First, one or two disks are placed at each vertex of the polygon. Second, holes in the polygon are connected to the boundary by adding disks tangent to two holes, or to a hole and the outer boundary. Third, disks are added to the as-yet-uncovered regions (called *remainder regions*), recursively reducing their complexity until all have at most four sides.

Disks at Corners. The first step preprocesses P so that we need only consider arc-gons with angle 0 at each vertex. At every convex vertex of P, we add a small disk tangent to both edges, as shown in Figure 2(a). At every concave vertex of P, we add two disks of equal radii, tangent to the edges, and tangent to the angle bisector at the corner, as shown in Figure 2(b). We can handle point holes by centering small disks at them. We choose radii small enough that disks lie within P, and none *overlap* (that is, intersect at interior points). This step isolates a small 3- or 4-sided remainder region at each corner of P. The large remainder region is an arc-gon of $2n + r = O(n)$ sides, where n is the number of vertices of P and r is the number of concave corners.

The first step can be implemented in time $O(n \log n)$ using the GVD of P. By checking the adjacencies of GVD cells, we can determine the nearest non-incident edge for each vertex v of P; one-eighth this distance gives a safe radius for the disks next to v. (Our implementation actually uses some other choices of radii to reduce output size.)

Connecting Holes. The second step connects polygonal holes to the outer boundary by repeatedly adding a disk tangent to two or more connected components of the boundary. In this step, previous disks touching a hole boundary are considered to be part of the hole. At the end, the large remainder region is bounded by a simply-connected arc-gon with $O(n)$ sides. Every corner of this arc-gon has angle 0, since each results from a tangency.

The second step can be implemented in time $O(n \log^2 n)$. We use a data structure that answers queries of the following form: given a query point p, which data object (vertex, edge, or disk) will be hit first by an expanding circle tangent to a vertical line through p (tangent at p and to the left of the line)? Such a query can be answered using Fortune's $*$-map [13], a sort of warped Voronoi diagram.

The initial set of data objects includes the edges, vertices, and disks attached to the outer boundary of the input polygon. The first query point is the leftmost point on a hole. The answer determines a disk D entirely contained within the polygon, touching both the hole and the outer boundary. Disk D is inserted into the query data structure, along with the vertices, edges and disks of the hole. Each subsequent query is performed using the leftmost point of all remaining holes. Altogether, the queries yield a set of disks connecting all holes and the exterior of the polygon.

For a static set of data objects, the $*$-map can be built in time $O(n \log n)$ [13], and standard planar subdivision search techniques [21] yield an $O(\log n)$ query time. In our case, the set of data objects is not fixed, since edges and a disk are added following each query. A trick due to Bentley and Saxe [5] allows dynamic insertions to the query structure, with query time $O(\log^2 n)$ and amortized insert time $O(\log^2 n)$. The trick is to divide the n data objects among $O(\log n)$ data structures, one for each bit in the binary representation. of n. A query searches all data structures in $O(\log^2 n)$ time. An insertion rebuilds all the data structures corresponding to bits that change. The total time required for n insertions is $O(n \log^2 n)$.

Reducing to 3- and 4-Sided Remainder Regions. After the first two steps, there is one simply-connected remainder region A with $O(n)$ sides, and $O(n)$ remainder regions in corners with three or four sides. The final step of the disk-packing stage recursively subdivides A by adding disks. The result will be a linear number of remainder regions of three and four sides.

To subdivide arc-gon A, we add a disk tangent to three of its sides. Such a disk divides the region enclosed by the arc-gon into four pieces: the disk itself and three smaller regions bounded by arc-gons. We choose a disk tangent to three sides of A, not all of them consecutive, thereby ensuring that each of the three smaller arc-gons has at most $n - 1$ sides. As shown in Figure 3, a disk tangent to three sides of an arc-gon must be centered

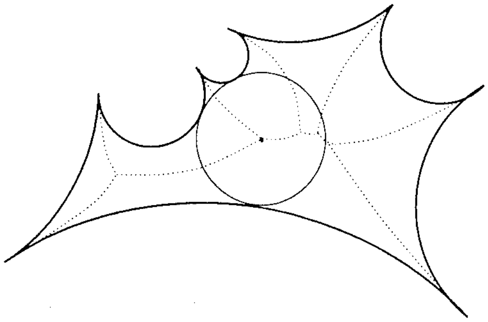


Figure 3. A disk tangent to three edges of an arc-gon is centered at a vertex of the GVD.

at a vertex of the GVD. Since A is simply connected, the edges of its GVD form a tree, a fact that will be useful in bounding the running time.

Lemma 1. *It is possible to reduce all remainder regions to at most 4 sides, by packing $O(n)$ non-overlapping disks into arc-gon A.*

Proof: Each vertex of the GVD corresponds to a disk tangent to three sides of A. If A has at least five sides, then there is a vertex v of the GVD that is adjacent to two non-leaf vertices of the GVD; a disk centered at v is tangent to three sides of A that are not all consecutive.

Now let $d(n)$ be the maximum number of disks needed to reduce an n-sided arc-gon to 3- and 4-sided remainder regions. We prove $d(n) \leq n-4$ by induction on n. The base cases are $d(3)=0$ and $d(4)=0$.

For the inductive step, notice that adding one disk produces three new arc-gons. (We can simply ignore extra tangencies in the degenerate case of four or more tangencies.) Suppose the new arc-gons have k, l, m sides, respectively, with $3 \leq k \leq l \leq m$. Since we are choosing non-consecutive sides, $m < n$. Counting 1 for the added disk, we have that $d(n) \leq 1 + d(k) + d(l) + d(m)$. Since the disk divides three sides, and is itself divided in three places, we have $k+l+m = n+6$.

First suppose $k = 3$. Since we are choosing non-consecutive sides, $l \geq 4$, so

$$\begin{aligned} d(n) &\leq 1 + d(3) + d(l) + d(m) \\ &\leq 1 + 0 + (l-4) + (m-4) \\ &= (l+m) - 7 = (n+3) - 7 = n-4. \end{aligned}$$

When $k \geq 4$, we have $d(n) \leq 1 + d(k) + d(l) + d(m)$. By induction, $d(n) \leq 1 + (k-4) + (l-4) + (m-4)$, which is equal to $(k+l+m) - 11 = (n+6) - 11 = n-5$. ■

Finally we comment on running time. Any tree contains a vertex, called a *centroid*, whose removal leaves subtrees of size at most two-thirds the original size. By choosing a disk centered at a centroid of the GVD of A, we split A into arc-gons A_1, A_2, and A_3. We imagine splitting A_1, A_2, and A_3 in parallel, so that altogether there will be $O(\log n)$ splitting stages, each involving a set of arc-gons of total complexity $O(n)$. If we recompute GVD's from scratch after each splitting stage, we obtain total time $O(n \log^2 n)$. This can be improved to $O(n \log n)$ by rebuilding GVD's in linear time, using an adaptation of the algorithm due to Aggarwal et al. [1].

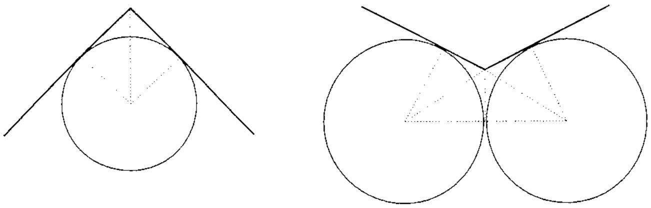


Figure 4. Remainder regions with vertices of P.

4. Triangulating the Pieces

We now describe the second stage of our algorithm. At this point, polygonal region P has been partitioned into disks and remainder regions with three or four sides, either straight or circular arcs. Each circular arc of a remainder region R is naturally associated with a pie-shaped sector, namely the convex hull of the arc and the center of the circle containing the arc. We denote the union of R and its associated sectors by R^+. These *augmented* remainder regions define a decomposition of P into simple polygons with disjoint interiors. In an augmented remainder region, we retain vertices at circle tangencies; vertices such as these, at which the angles measure π, are called *subdivision points*.

In this section, we show how to triangulate each R^+ region. All Steiner points will lie either on straight sides of R (that is, along P's boundary) or interior to R^+. Thus we never place Steiner points on the radii bounding sectors, and triangulated R^+ regions will fit together at the end. Our triangulation method is given in three cases: remainder regions with vertices of P, three-sided remainder regions, and four-sided remainder regions. The first two cases are easy, but the last is quite intricate. In all cases, triangulating a single R^+ region takes $O(1)$ time, so altogether the running time of the second stage is $O(n)$.

Remainder Regions with Vertices of P. Every vertex of P was isolated by one or two disks in the first step of the algorithm. The resulting regions R^+ can be triangulated with at most four right triangles, as shown in Figure 4, by adding edges from the disk centers to the points of tangency and the vertex of P.

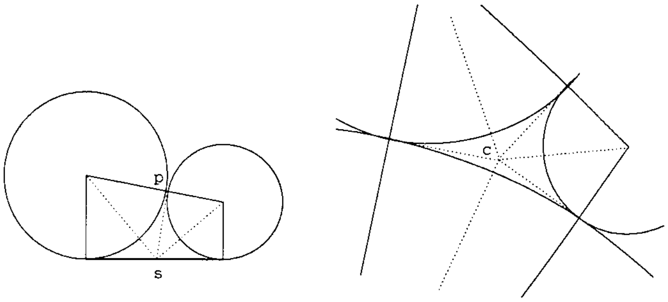


Figure 5. Three-sided remainder regions: (a) with a straight side, (b) with only finite-radius arcs.

Three-Sided Remainder Regions. A three-sided remainder region R without a vertex of P is bounded by three circular arcs, so that arcs meet tangentially at the vertices of R. Here we are considering a straight side to be an arc of an infinitely large circle. We call a Steiner point in an augmented remainder region R^+ *safe* if it lies either interior to R^+ or on the boundary of P.

Lemma 2. *If R is a three-sided remainder region, then R^+ can be triangulated with at most six right triangles, adding only safe Steiner points.*

Proof: First assume that R has a straight side (necessarily at most one), and view R so that this straight side forms a horizontal base. The augmented region R^+ is a trapezoid with two vertical sides, and a subdivision point p along its slanted top side. We cut perpendicularly from p (that is, tangent to both arcs) across R until we hit the base, and there add a safe Steiner point s. We add edges from s to the centers of the arcs' circles to divide R^+ into four right triangles, as shown in Figure 5(a).

Now assume all the sides of R are arcs of finite radius. Notice that R^+ is a triangle with subdivided sides. Moreover, the subdivision points along the sides of R^+ are exactly the tangency points of the inscribed circle of R^+. (This follows from the fact that the inscribed circle makes each corner of R^+ incident to two edges of equal length.) So we add the circle's center c and edges from c to all the vertices around R^+, dividing R^+ into six right triangles, as shown in Figure 5(b). ∎

Four-Sided Remainder Regions. A four-sided remainder region R is bounded by four circular arcs (possibly of infinite radius) that meet tangentially at the vertices of R. Lemma 3 states two interesting properties of these regions.

Lemma 3. *The arcs of R have total measure 2π. The vertices of R are cocircular.*

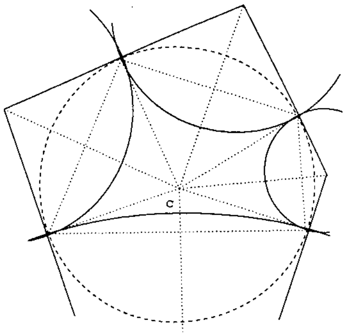


Figure 6. The good case for four-sided remainder regions.

Proof: If all arcs have finite radius, then the sum of the measures of the arcs of R is identical to the sum of the measures of the angles at the corners of R^+. For straight sides, we imagine further augmenting R^+ with "infinite sectors" of angle 0.

Next we show that the vertices are cocircular. Let C_1 and C_3 be finite-radius circles containing opposite arcs of R. (Here notice that if R has two straight sides, they must be opposite.) Assume the two lines that are externally tangent to both C_1 and C_3 meet at a point x. There exists an inversive transformation ([11], pp. 77–95) of the projective plane that maps x to infinity and hence the two external tangent lines to parallel lines. The transformed circles C_1' and C_3', corresponding to C_1 and C_3, have equal size, so the vertices of the transformed remainder region R' form an isosceles trapezoid. It is easy to see that any isosceles trapezoid has cocircular vertices. The inverse of the original inversive transformation maps the circle containing the vertices of R' to a circle containing the vertices of R. ∎

Now if we are lucky, the region R^+ can be triangulated with 16 right triangles, as in the following lemma.

Lemma 4. *If R is a four-sided remainder region, in which each arc measures at most π and the center of the circle through R's vertices lies in the convex hull of R, then R^+ can be triangulated with 16 right triangles, adding only safe Steiner points.*

Proof: We may assume that all arcs of R have finite radius. If R has a straight edge, we can apply the triangulation to a region with an infinite sector attached to the straight edge and then simply remove the resulting infinite strips.

The construction is shown in Figure 6. Here we have added the center c of the circle through R's vertices in order to form four *kites* (quadrilaterals with two adjacent pairs of equal-length sides). ∎

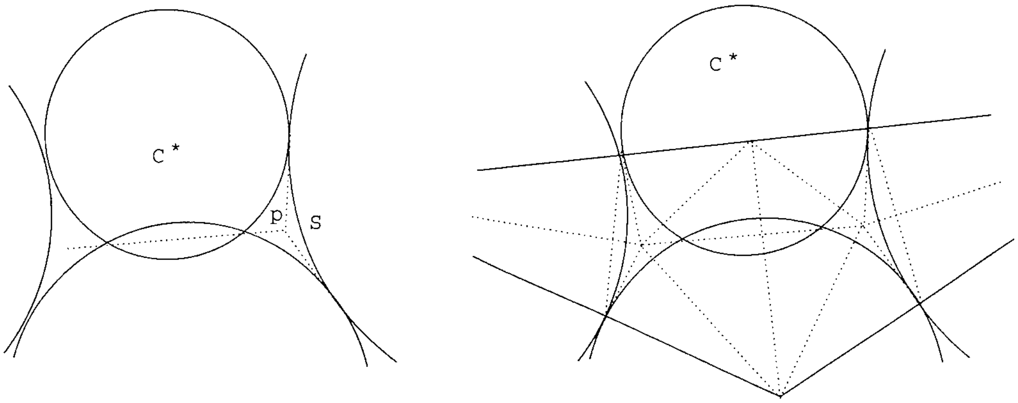


Figure 8. (a) Mutual tangents and mutual chord meet at a point. (b) Triangulation.

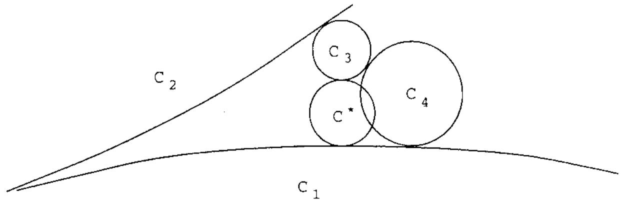


Figure 7. New circle C^* breaks up a reflex remainder region.

The triangulation of Figure 6 can fail in two different ways: (1) if one of the arcs of R measures more than π (a *reflex* arc), then R^+ has a reflex vertex at which angles will measure more than $\pi/2$; and (2) if center c lies outside the convex hull of R, then it lies on the wrong side of one of the chords and will introduce unwanted intersections. Each of these difficulties will be handled by adding yet another disk.

First assume R has a reflex arc on circle C_3. Add another disk C^*, tangent to C_3 and C_1 (the circle containing the arc opposite C_3), such that the center of C^* lies on the line joining the centers of C_1 and C_3. The new disk C^*—unlike any of the disks used up until this point—may overlap an old disk and produce a self-intersecting remainder region, as shown in Figure 7. Lemma 3 holds without modification for self-intersecting remainder regions. Region R^+, formed as before by adding the associated sectors to R, remains a simple polygon with subdivision points on its sides, specifically a triangle with three subdivisions on one side and one on each of the others. The next lemma shows how to triangulate R^+ with a generalization of the method of Lemma 2.

Lemma 5. *Let R be a self-intersecting four-sided remainder region resulting from breaking up a reflex four-sided remainder region by the addition of C^*. Then R^+ can be triangulated with at most 12 right triangles, adding only safe Steiner points.*

Proof: Again we may assume that all arcs of R have finite radius, as a solution to this case implies a triangulation for the case of straight sides.

Consider one of the arcs S next to C^*. We claim that the lines tangent to S at its endpoints and the mutual chord of C^* and its opposite arc all meet at a single point p interior to R, as shown in Figure 8(a). This claim allows the triangulation shown in Figure 8(b).

Why is the claim true? For each of the three disks—C^*, the opposite disk, and the one with arc S—we define a *power* function. The power function of a circle with center (x_c, y_c) and radius r is $P(x,y) = (x-x_c)^2+(y-y_c)^2-r^2$. The power functions of two tangent circles are equal along their mutual tangent line; the power functions of two overlapping circles are equal along a line containing their mutual chord. The point p of the claim is the point at which all three power functions are equal. ■

We now consider the second difficulty. Call a four-sided remainder region R *centered* if the convex hull of R contains the center c of the circle through R's vertices, and *uncentered* otherwise. Let the arc of R with the longest chord lie along circle C_1, and denote the other circles by C_2, C_3, and C_4, clockwise around R. (Circles through infinity handle the case of straight sides.) Assume that the line through the centers of C_1 and C_3 is vertical as in Figure 9. If R is uncentered, then c must lie below the chord on C_1.

Let t_{12} be the vertex of R at which C_1 and C_2 meet, and similarly define t_{23}, t_{34}, and t_{41}. For a disk C^* tangent to both C_1 and C_3, let S_L $(= S_L(C^*))$ be the circular arc with endpoints t_{12} and t_{23} that passes through the points at which C^* meets C_1 and C_3. Lemma 3 guarantees that such an arc exists. Similarly define S_r. Let c_L and c_r be the centers of the circles containing S_L and S_r, respectively.

Lemma 6. *There exists a disk C_c^* tangent to C_1 and C_3, such that c_L lies in the convex hull of the four points of tangency around S_L and c_r lies in the convex hull of the four points of tangency around S_r.*

Proof: First let C^* be the disk that is tangent to C_1 and C_3 such that the center of C^* lies on the line through the centers of C_1 and C_3.

Centers c_L and c_r lie on a horizontal line through the center of C^*, hence outside C_1 and C_3. But the requirements of the lemma may be violated, because c_r may lie outside the chord $t_{34}t_{41}$ of S_r (if S_r has measure less than π) or c_L may lie outside the chord $t_{12}t_{23}$ of S_L (if S_L has measure less than π). We assert that both of these bad conditions cannot occur at the same time. Why? It suffices to show that the sum of the measures of S_L and S_r is at least 2π. Angle $\angle t_{23}t_{*3}t_{34}$, where t_{*3} is the point of tangency of C^* and C_3, measures at least $\pi/2$, because the arc of R on C_3 measures at most π. And $\angle t_{12}t_{*3}t_{41}$ measures more than $\pi/2$, because the center of the circle through the vertices of R lies below $t_{12}t_{41}$. Hence the remaining angles at t_{*3} (the two subtended by the endpoints of S_L and S_r) sum to less than π.

If neither bad condition occurs, then C^* satisfies the conditions of the lemma, and we are done. But if one of the bad conditions does occur, then we sweep C^* in the direction that could cure the condition, while keeping C^* tangent to both C_1 and C_3. If c_r lies outside $t_{34}t_{41}$, then we sweep C^* to the left in Figure 9; the other case is symmetrical.

During the leftward sweep, c_r moves towards C_1 along the perpendicular bisector of $t_{34}t_{41}$ and c_L moves towards C_2 along the perpendicular bisector of $t_{12}t_{23}$, as shown in Figure 9. These bisectors never intersect C_3, so c_L and c_r can never lie outside their chords ($t_{23}t_{*3}$ and $t_{*3}t_{34}$) on C_3. The chords of S_L and S_r between tangency points on C^* are never the longest chords on these arcs, so c_L and c_r also lie safely inside these chords throughout the sweep.

As c_r moves, it must pass through $t_{34}t_{41}$ and become good, before it reaches C_1 and becomes bad. By the arc-measure argument above, c_r must cross inside $t_{34}t_{41}$ before c_L crosses outside $t_{12}t_{23}$. Hence at some point in the sweep, both c_r and c_L satisfy the conditions of the lemma, and the C^* at this point is C_c^*. ∎

Lemma 6 breaks up uncentered, non-reflex remainder regions, but unless C_c^* coincides with the initial C^* in the sweep, adding C_c^* creates a new reflex remainder region. The following lemma finesses this final difficulty (shall we say circularity?) by triangulating both new augmented regions at once.

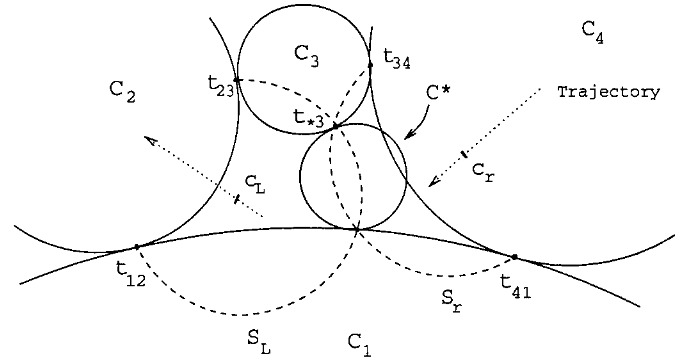


Figure 9. The trajectories of centers c_L and c_r as C^* sweeps.

Lemma 7. *Let R be a non-reflex, uncentered, four-sided remainder region. Then R^+ can be triangulated into at most 28 right triangles, adding only safe Steiner points.*

Proof: Again we may assume that all arcs of R have finite radius. We start by adding the "centering" disk C_c^*, guaranteed by Lemma 6. As above, we denote the tangent point of C_c^* and C_1 by t_{*1} and the tangent point of C_c^* and C_3 by t_{*3}. In addition to t_{*1} and t_{*3}, we add the following Steiner points: the centers c_ℓ and c_r associated with arcs S_ℓ and S_r, and the midpoint m of segment $t_{*1}t_{*3}$. See Figure 10.

We triangulate by adding: all chords around S_ℓ and S_r; edges from c_ℓ to m and to the centers of C_1, C_2, and C_3; and edges from c_r to m and to the centers of C_3, C_4, and C_1. Finally we add an edge between the center of C_1 and t_{*1} and between the center of C_3 and t_{*3}.

Resulting triangles come in sets of four, each set forming a kite and its diagonals. Hence all triangles are right. (Notice that C_c^* is treated somewhat differently than the other circles, since we do not use its center. Nevertheless the four triangles around m form a kite, because $t_{*1}t_{*3}$ is the mutual chord of C_c^*, S_2, and S_4.) ∎

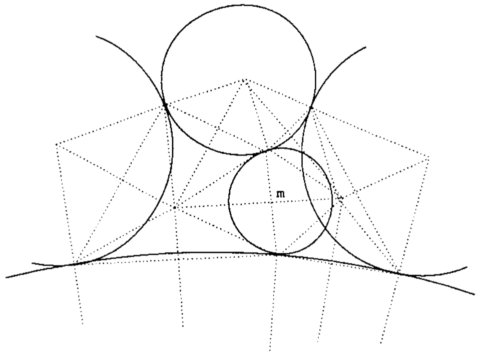


Figure 10. Triangulating R^+ when R is uncentered.

We have now completed the proof of our main theorem, linear-size nonobtuse triangulation.

Theorem 1. *An n-vertex polygonal region can be triangulated with $O(n)$ right triangles, in time $O(n \log n)$ for simple polygons and $O(n \log^2 n)$ for polygons with holes.* ∎

5. Implementation

We implemented our algorithm within the Matlab environment [15]. The implementation differs somewhat from the algorithm described in the text. We use several heuristics for disk placement so as to reduce the number of triangles. Also we do not bother to compute generalized Voronoi diagrams. Rather we use a simple $O(hn)$ method to connect h holes to the boundary, and we choose arbitrary disks touching three non-consecutive sides, rather than disks centered at GVD centroids. To keep the user entertained during the worst-case $O(n^2)$ running time, we display color-coded disks and triangles as they are added. Finally, although Section 4 describes a construction using only right triangles, the implementation produces some acute triangles, an example being the large downward-pointing triangles in Figure 1(c).

Experiments with a variety of polygonal regions show that an n-vertex input typically produces about $22n$ triangles. A simple polygon with $n-3$ reflex corners can produce as many as $25n$ triangles; the maximum for polygons with holes appears to be about $33n$. Since a floating point representation entails roundoff, some of the right angles present in the nonobtuse triangulation become slightly obtuse. The worst test case had an angle of about $\frac{\pi}{2} + 10^{-11}$ radians (Matlab retains 16 digits), so the implementation is fairly robust, which is somewhat surprising given that our implementation often places very small disks next to very large ones.

6. Parallelizing the Algorithm

We now sketch the first $\mathcal{NC}$ algorithm for nonobtuse triangulation. We give a straightforward though rather inefficient algorithm, with parallel time $O(\log^3 n)$ and processor requirement $O(n^2)$. Both time and processors should be improvable. One bottleneck subproblem is the computation of the GVD of circular arcs; see [14] for the GVD of line segments.

Theorem 2. *An n-vertex polygonal region P (with holes) can be triangulated with $O(n)$ right triangles in $O(\log^3 n)$ time on $O(n^2)$ EREW PRAM processors.*

Proof: Using $O(n^2)$ processors—one for each vertex-edge pair—and time $O(\log n)$, we can compute the nearest non-incident edge for each vertex and hence choose appropriate radii for disks to pack into corners. The second step, connecting holes, is trickier. We first compute a minimum spanning tree (MST) of P's holes; by this we mean the shortest set of line segments S, each segment with both endpoints on the boundary of P, such that the union of S and the exterior of P is a connected subset of the plane. Using $O(n^2)$ processors and time $O(\log n)$, we compute for each vertex the nearest edge lying on a different connected component of P's boundary. We use this information to compute distances between connected components, and add to S the shortest component-joining line segment incident to each component. This reduces the number of components by at least a factor of two, so $O(\log n)$ such merging steps suffices to complete the computation of set S.

Now it is not hard to show that no point of the plane is covered by more than $O(1)$ diameter disks of segments in S. Hence there is a pairwise-disjoint set of diameter disks of cardinality a constant fraction of $|S|$ [26]. It is not hard to find these disks in parallel time $O(\log n)$ using separators. We repeat the process of computing the MST (of the new connected components, holes plus disks) and finding a large independent set of diameter disks. After $O(\log n)$ cycles—for total time of $O(\log^3 n)$—we have reduced to a simply-connected arc-gon.

The third step of the disk-packing stage uses the generalized Voronoi diagram in order to find centroid disks. Using $O(n^2)$ processors and time $O(\log^2 n)$, we can compute the GVD of a set of n circular arcs as follows. We compute the equal-distance curve (bisector) for each pair of arcs. Then for each arc a, we compute the piecewise-polynomial boundary of a's cell recursively by dividing the set of bisectors into two equal halves and then merging the boundaries for each half. Two piecewise-polynomial boundaries of $O(n)$ pieces can be merged in time $O(\log n)$ on n processors. Once, the GVD has been computed, a centroid can be found in time $O(\log n)$ by alternately removing leaves and merging degree-2 paths.

Recall that the algorithm requires a "decomposition tree" of centroid disks of height $O(\log n)$, so by simply recomputing the GVD after each centroid, we obtain an overall time for the third disk-packing step of $O(\log^3 n)$. Finally, the triangulation stage consists entirely of local operations, so it is trivially parallelized. ∎

7. Conclusion

We have presented a new algorithm for nonobtuse triangulation of polygons with holes. The number of triangles produced is linear in the number of vertices of the input, a significant improvement over previous methods. This is of course asymptotically optimal, resolving the question of the theoretical complexity of nonobtuse triangulation of polygons.

One direction for further work is extending the algorithm to inputs more general than polygons with holes; these inputs occur in modeling domains made of more than one material. Currently, there is an algorithm for refining a triangulated simple polygon into a nonobtuse triangulation with $O(n^4)$ triangles, and also an $\Omega(n^2)$ lower bound [8]. There is still no algorithm for polynomial-size nonobtuse triangulation of planar straight-line graphs; a solution to this problem would give another solution to "conforming Delaunay triangulation" [12]. Mitchell [17] recently showed how to triangulate planar straight-line graphs with maximum angle at most $7\pi/8$, using at most $O(n^2 \log n)$ triangles.

Another important direction is exploring whether our ideas can be used for related mesh-generation problems. For instance, disk-packing may yield a simpler algorithm for the problem of no-small-angle, nonobtuse triangulation [3, 16]. Perhaps we can use our methods to produce nonobtuse meshes with skinny triangles aligned with the boundary. (See [19] for aligned no-large-angle meshes.) Or perhaps our methods can be allied with a heuristic method called "bubble systems" [24].

Finally, higher dimensions are still a mystery. Do three-dimensional polyhedra admit polynomial-size triangulations without obtuse dihedral angles? Algorithms for point sets are known [6, 10].

Acknowledgments

We would like to thank Paul Chew, Jonathan Shewchuk, and Shang-Hua Teng for some valuable discussions and Dan Asimov and Micha Sharir for proofs of Lemma 3.

References

[1] A. Aggarwal, L.J. Guibas, J. Saxe, and P.W. Shor. A linear time algorithm for computing the Voronoi diagram of a convex polygon. *Disc. and Comp. Geometry* 4 (1989), 591–604.

[2] I. Babuška and A. Aziz. On the angle condition in the finite element method. *SIAM J. Numer. Analysis* 13 (1976), 214–227.

[3] B.S. Baker, E. Grosse, and C.S. Rafferty. Nonobtuse triangulation of polygons. *Discrete and Comp. Geom.* 3 (1988), 147–168.

[4] R.E. Bank. *PLTMG User's Guide.* SIAM, 1990.

[5] J.L. Bentley and J.B. Saxe. Decomposable searching problems: 1. Static-to-dynamic transformation. *J. Algorithms* 1 (1980), 301–358.

[6] M. Bern, L.P. Chew, D. Eppstein, and J. Ruppert. Dihedral Bounds for Mesh Generation in High Dimensions. Manuscript, 1993.

[7] M. Bern, D. Dobkin, and D. Eppstein. Triangulating polygons without large angles. *Proc. 8th Annual ACM Symp. Computational Geometry*, 1992. To appear in *IJCGA.*

[8] M. Bern and D. Eppstein. Polynomial-size nonobtuse triangulation of polygons. *Int. J. Comp. Geometry and Applications* 2 (1992), 241–255.

[9] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. Xerox PARC Tech. Report CSL-92-1. Also in *Computing in Euclidean Geometry*, World Scientific, Singapore, 1992.

[10] M. Bern, D. Eppstein, and J.R. Gilbert. Provably good mesh generation. *Proc. 31st IEEE Symp. on Foundations of Computer Science*, 1990, 231–241. To appear in *J. Comp. System Science.*

[11] H.S.M. Coxeter. *Introduction to Geometry.* John Wiley & Sons, 1961.

[12] H. Edelsbrunner and T.S. Tan. An upper bound for conforming Delaunay triangulations. *Discrete and Comp. Geom.* 10 (1993), 197–213.

[13] S. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica* 2 (1987), 153–174.

[14] M.T. Goodrich, C. ÓDúnlaing, and C. Yap. Computing the Voronoi diagram of a set of line segments in parallel. *Algorithmica* 9 (1993), 128–141.

[15] *MATLAB Reference Guide*, The MathWorks, Inc., Natick, Massachusetts, 1992.

[16] E. Melissaratos and D. Souvaine. Coping with inconsistencies: a new approach to produce quality triangulations of polygonal domains with holes. *Proc. 8th Annual ACM Symp. Computational Geometry*, 1992, 202–211.

[17] S.A. Mitchell. Refining a triangulation of a planar straight-line graph to eliminate large angles. *Proc. 34th Symp. on Foundations of Computer Science,* 1993, 583–591.

[18] S.A. Mitchell. Finding a covering triangulation whose maximum angle is provably small. *Proc. 17th Annual Computer Science Conference*, *Australian Comp. Science Comm.* 16 (1994), 55–64.

[19] J.-D. Müller. Proven angular bounds and stretched triangulations with the frontal Delaunay method. *Proc. 11th AIAA Comp. Fluid Dynamics*, Orlando, 1993.

[20] S. Müller, K. Kells, and W. Fichtner. Automatic rectangle-based adaptive mesh generation without obtuse angles. *IEEE Trans. Computer-Aided Design* 10 (1992), 855–863.

[21] F. Preparata and M. Shamos. *Computational Geometry – an Introduction.* Springer-Verlag, 1985.

[22] J.F. Randolph. *Calculus and Analytic Geometry.* Wadsworth, 1961, 373–374.

[23] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. *Proc. 4th ACM-SIAM Symp. on Discrete Algorithms*, 1993, 83–92.

[24] K. Shimada and D.C. Gossard, Computational methods for physically-based FE mesh generation. *Proc. IFIP TC5/WG5.3 8th Int. Conference on PROLAMAT*, Tokyo, 1992.

[25] G. Strang and G.J. Fix. *An Analysis of the Finite Element Method.* Prentice Hall, 1973.

[26] S.-H. Teng. *Points, Spheres, and Separators: a unified geometric approach to graph partitioning.* Ph.D. Thesis, Carnegie Mellon University, CMU-CS-91-184, 1991.

[27] S.A. Vavasis. Stable finite elements for problems with wild coefficients. Tech. Report TR93-1364, Dept. of Computer Science, Cornell University, 1993.

Bounds on the Size of Tetrahedralizations

BERNARD CHAZELLE*
Department of Computer Science
Princeton University
Princeton, NJ 08544, USA

NADIA SHOURABOURA†
Program in Applied and Comput. Math.
Princeton University
Princeton, NJ 08544, USA

1 Introduction

We extend the work of Bern [5] on triangulating the region between three-dimensional polyhedra. Here is a summary of our results:

- We give a linear algorithm for triangulating the region between two convex polyhedra. No assumptions are made on the polyhedra: they can be disjoint, nested, or overlapping. The algorithm produces $O(n)$ tetrahedra, where n is the input size. This optimal bound improves on an $O(n \log n)$-size construction of Bern [5].
- We give a method for triangulating the region between a convex polyhedron and a disjoint polyhedral terrain (or a star-shaped polyhedron). The number of tetrahedra produced is $O(n \log n)$. We prove the rather surprising result that the bound is tight. More generally, we show that any convex decomposition of the space between a convex polyhedron and a terrain requires $\Omega(n \log n)$ pieces in the worst case.
- We prove that any polyhedron of genus g must have at least $g-1$ reflex dihedral angles. To our surprise, this result appears to be new. Actually, we were not even able to find any proof in the literature that the number of reflex edges is $\Omega(g)$. Although our proof is quite simple it relies on the Gauss-Bonnet curvature formula and thus it is not completely self-contained. This result shows that the triangulation algorithm of Chazelle and Palios [8] works just the same for polyhedra of arbitrary genus: specifically, any polyhedron with n vertices and r reflex angles can be triangulated with $O(n + r^2)$ tetrahedra, regardless of its genus. The bound is tight in the worst case.

Tetrahedralization refers to the triangulation of three-dimensional polyhedra or the region between them. The subject has been heavily researched because of its relevance to the finite-element method, mesh generation, topology of 3-manifolds, interpolation theory, tool design, etc. [1, 2, 3, 4, 5, 8, 9, 14, 15]. An odd assortment of results has emerged. For example, all polyhedra can be tetrahedralized, but nonconvex ones might require additional vertices, so-called Steiner points: the canonical example is the Schönhardt octahedron [17], which is made by connecting together two parallel, slightly twisted triangles. Checking whether Steiner points are needed or not has been shown to be NP-hard by Ruppert and Seidel [16]. Steiner points are somewhat of a nuisance in practice, because they make representations more complicated and cause round-off error problems. Unfortunately they can rarely be avoided. The number and the shape of the tetrahedra are parameters that are both more critical and malleable. The former is where we focus our attention in this paper.

It was shown in [8] that any n-vertex polyhedron can be triangulated using only $\Omega(n^2)$ tetrahedra. This is known to be optimal in the worst case [6]. Of course, in many cases, it is possible to do much better. For example, a convex polyhedron requires only $O(n)$ tetrahedra. Bern [5] provided motivation for looking at the region between two convex polyhedra. If the polyhedra are disjoint (side-by-side) it is quite easy to achieve an $O(n)$-size triangulation. The nested case is more difficult, and Bern proposed a construction consisting of $O(n \log n)$ tetrahedra. Our first result improves this to $O(n)$, which is optimal. Furthermore, the method allows the polyhedra to be intersecting. It is based on a dovetailed construction of the Dobkin-Kirkpatrick polyhedral hierarchy [10], somewhat in the style of [7] though much simpler. Unlike Bern's construction, our method uses Steiner points. Whether this can be avoided remains an open problem.

*Supported in part by NSF Grant CCR-90-02352 and The Geometry Center, University of Minnesota, an STC funded by NSF, DOE, and Minnesota Technology, Inc.

†Supported in part by NSF Grant PHY-90-21984

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

The polyhedron used in the quadratic lower bound proof of [6] is built by placing two polyhedral terrains,[1] one on top of the other. It is natural to ask whether the lower bound collapses if one terrain is convex. Our second result shows that $\Theta(n \log n)$ is the exact answer. The upper bound is obtained by carefully pruning the outer polyhedral hierarchy at critical places and tracing its interaction with the terrain by way of a fairly delicate accounting scheme.

The lower bound is achieved by creating an art-gallery that requires on the order of $n \log n$ guards: the roof of the gallery is a hangar-like concave structure; the floor plan is a terrain parameterized by a permutation. We prove that the number of guards needed is precisely equal to the number of swaps required in *mergesorting* that permutation. Expectedly, we use the bit-reversal permutation to achieve the best possible lower bound. The essence of our proof technique is to be able to interpret a triangulation scheme as a permutation routing network, whereby each node corresponds to one or several distinct tetrahedra. This connection allows us to classify the floor plans that lead to easy triangulations by just looking at the complexity of their corresponding permutations.

2 Triangulating between Two Convex Polytopes

Let P and Q be two convex polytopes,[2] with a total of n vertices. We wish to subdivide the region between the boundaries of P and Q into $O(n)$ tetrahedra. Naive methods such as merging the Dobkin-Kirkpatrick hierarchies of the two polytopes are doomed to fail. Instead, we define a process for "thickening" the boundaries of the polytopes in coordinated fashion, until they become so wide as to cover all of $\mathbb{R}^3$: during the growth process we maintain a triangulation of the union of the thickened boundaries. For this to work, the two boundaries have to grow at comparable speeds.

To flesh out this idea, we introduce the key notion of a *polyhedral layer*, which is simply the region between two nested convex polytopes. The outer and inner boundaries might share faces, so a layer is topologically equivalent to $S^2 \times [0, 1]$ pinched at various places. We define the *magnitude* of a layer to be the total number of faces (of all dimensions) among its inner and outer boundaries. Layers will always come along with a cell decomposition, so any standard data structure for three-dimensional cell complexes[3] will provide an adequate representation [11]. Specifically, we need lists of vertices, edges, facets, and cells, along with their incidence relations.

Thickening the Boundaries. A layer can grow inwards or outwards. We begin with the inward growth. Pick a maximal independent set of low-degree vertices on the inner boundary. For each such vertex v, form the convex hull of its adjacent vertices and keep only those faces non adjacent to v: glue the resulting patch to the layer. The space between the patch and the layer is called a *pocket* (Fig. 1). It is a polyhedron of constant description size. Growing a convex polytope inwards in this fashion produces the well-known Dobkin-Kirkpatrick hierarchy [10].

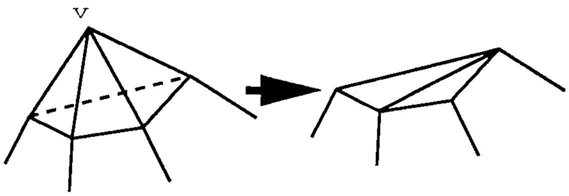


Figure 1.

The outward growth [12, 13] is equivalent to the inward growth in dual space. We select a maximal independent set of nonadjacent, low-complexity facets on the outer boundary of the layer. For each such facet f, we intersect the halfspaces bounded by f and its adjacent facets (on the relevant sides). This creates a pocket with at most a constant number of other facets, one of which is f (Fig. 2). There is one delicate point to address: the facets of the pocket are coplanar with the facets adjacent to f. On the one hand, we must merge those facets together, otherwise the complexity of the new outer boundary does not decrease (actually it strictly increases). On the other hand, if we merge

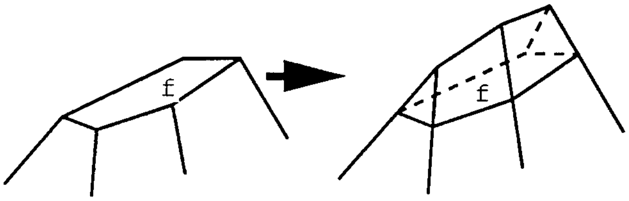


Figure 2.

[1] A polyhedral terrain is the graph of a bivariate piecewise-linear continuous function: any vertical line intersects it in exactly one point.

[2] A polytope is a bounded polyhedron: our restriction to the bounded case is meant only to simplify the explanation. It is routine to extend our results to unbounded polyhedra. Similarly, we assume that the polytopes are nondegenerate and well-behaved. In particular, it is assumed that no vertex is completely surrounded by coplanar facets: if this were the case, we would remove the vertex and merge the incident facets together.

[3] Recall that a cell complex is a subdivision of a region into relatively open faces (i.e., vertices, edges, facets, cells) such that the closures of two adjacent faces intersect in the closure of a face.

the facets, we no longer have a cell complex. To resolve this dilemma, we enclose the new outer boundary into a slightly enlarged copy of itself, and connect it in the obvious way (Fig. 3). This creates a layer of very thin *buffer* pockets that completely surrounds the outer boundary; such a pocket is a symbolic device and need not have actual volume.

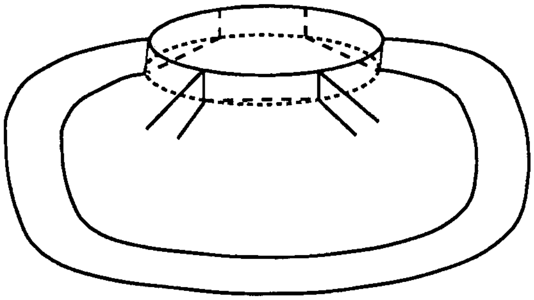


Figure 3.

A simple but important observation is that thickening both the inner and outer boundaries of the layer decreases its magnitude by a constant factor. Note that the magnitude of a layer may have nothing to do with its complexity as a cell complex: as a matter of fact, growing a layer increases the complexity of its cell complex but decreases its magnitude.

Coordinated Growth of Two Layers. Let us now look at the thickening of two layers, L and L', emanating from P and Q, respectively. We do not make any assumptions about the relative positioning of the layers. We assume that the union of the two layers is fully available as a cell complex (not necessarily simplicial); we call it the *union complex.* Note that it also provides a cell decomposition of the boundary of $L \cup L'$. We do not assume, however, that it gives a cell decomposition of the four boundaries. For example, portions of the boundaries strictly inside $L \cup L'$ might have been removed and have no representative features in the union complex.

We assume that all four boundaries of the layers are available separately, i.e., with no regard to the fact that some might intersect: this is the same as having a representation of four distinct convex polytopes. However, we also provide pointers from the facets of the four boundaries to their corresponding facets of the union complex: for this to make sense, we require that any boundary facet that does not lie completely inside $L \cup L'$ should have its "outside" part appear as the union of at most a constant number of facets of the union complex. (Just one is not enough, because such a facet might be homeomorphic to an annulus, so it needs to be cut up in two in order to keep all the facets topologically equivalent to disks.)

The reason we need this correspondence will now become clear. Suppose that we wish to grow, say, the layer L inwards. Our first task is to determine the intersection of the new pockets of L with the current union complex. If a pocket has a facet in (old) ∂L that intersects L' then routine navigation through the cell complex starting at that intersection does the job. If not, then things are a little more dicey, as the pocket might penetrate L' at some random location. We use the optimal algorithm of [7] to intersect the boundaries of L and L' pairwise. With the correspondence mentioned above, this gives us access to all the union complex boundary facets that intersect the pockets, in time linear in the complexity of the boundaries. (There are technical details which we omit.)

What do we do with the new pockets? A pocket that does not intersect either boundary of L' falls in one of two categories (Fig. 4): if it lies entirely inside L', it is ignored, i.e., it has no effect on the union complex. Otherwise, it is simply added to the union complex. The more interesting case arises if a pocket partly intersects L' (Fig. 5). Then, we triangulate the complement of L' within the pocket and add these faces to the current union complex. In Fig. 5 this complement consists of two full-dimensional components; in general, it can have up to a constant number of them (because the pocket has $O(1)$ complexity). We should also observe that the union complex is only modified by growing: once it contains a face it keeps it forever. Also, as we mentioned before, we should not expect the boundary of the new layer L to be represented in the union complex in its totality.

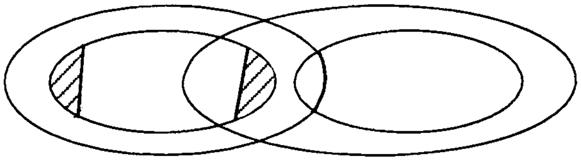


Figure 4.

Triangulating the complement of L' in the pocket is quite easy because the boundary of L' is convex and the pocket has constant size. Outward growth is handled similarly. The only difference is that a pocket might now have a large size (remember those thin buffer pockets). It is an easy exercise, however, to carry out the triangulation in linear time.

The Triangulation Algorithm. Returning now to our original problem, we grow the layer of P first inwards and then outwards. Then, we switch and do the same with respect to Q. We iterate this process

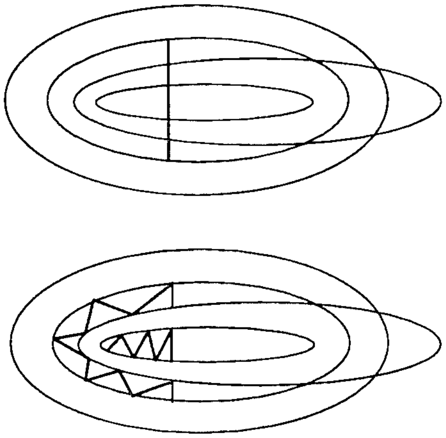


Figure 5.

back and forth until the boundaries become of constant size. At that point, a simple finishing touch adds a few unbounded tetrahedra to create a full-fledged union complex that subdivides all of $\mathbb{R}^3$. Buffer pockets that have not yet been triangulated can be handled now, which produces the desired simplicial complex. The total space and time complexity is easily seen to follow a geometric series summing up to $O(n)$.

Theorem 2.1 *It is possible to triangulate the region between the boundaries of two convex polytopes in linear time, using a linear number of faces.*

Remark: The dovetailing mechanism can be replaced by variants such as always thickening the bigger layer. However, it would be a big mistake to grow one layer all the way, and then work on the other. Similarly, simply merging the two hierarchies fails miserably. In both cases, on the order of $n \log n$ tetrahedra might end up being produced. As one should note, a key idea to avoid this blowup has been to avoid refining the interior of the layers as they grow.

3 Triangulating between a Polytope and a Terrain

Let P and Q be respectively a convex polytope and a polyhedral terrain. We could also choose an arbitrary star-shaped polyhedron as Q, since it can be made into a terrain by a suitable projective transformation. We assume that P and Q do not intersect, or if they do, that the input size n accounts for all the intersection features as well.

Theorem 3.1 *It is possible to triangulate the region between two disjoint polytopes (one convex, the other star-shaped), using $O(n \log n)$ tetrahedra, where n is the size of the input polyhedra. The size of the triangulation is asymptotically optimal in the worst case.*

3.1 The Upper Bound

A natural idea would be to compute the outer Dobkin-Kirkpatrick hierarchy of P and merge the terrain Q within it. Such a naive approach runs into problems, however, because terrain facets may intersect too many features of the hierarchy. We get around this problem by pruning the resulting cell complex appropriately. Recall from the previous section that the outer hierarchy of P subdivides $\mathbb{R}^3$ into constant-complexity pockets and arbitrarily thin buffer pockets. Suppose that two facets f, g of the terrain intersect the same pocket and:

1. Both facets intersect the same set of pocket edges.
2. No edge of either facet intersects the pocket.
3. The portion D of the pocket between the two facets does not intersect the terrain (except at f, g).

A polytope D that satisfies all these conditions is called a *drum* (Fig. 6): it consists of two facets (which are pieces of the terrain facets f, g that cut all across the pocket) and at most a constant number of lateral facets (pieces of pocket facets): the pair (f, g) constitutes the *type* of the drum. We say that two drums are adjacent if they share a common lateral facet. Our pruning strategy involves removing all the lateral facets between adjacent drums (along with the incident edges and vertices, unless they are needed by neighboring non-drum cells). Note that because two adjacent drums are always of the same type, pruning never brings in contact drums of different type. The merged drums are called *superdrums.* Two obvious questions are: How do we turn the resulting subdivision into a triangulation? Why is the triangulation of size $O(n \log n)$?

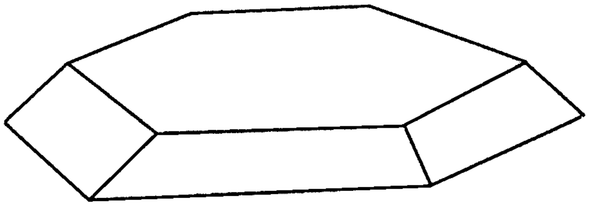


Figure 6.

Completing the Triangulation. Let us examine each pocket separately. If the pocket is free of any terrain features, then triangulating it is routine (as usual, one must be careful that the triangulations of a facet should agree on both sides, but this is easy to enforce;

see [5] for a thorough treatment of a more general and difficult version of that issue.) If the pocket does not give rise to any drum, then no pruning takes place within and we must triangulate a portion of the terrain inside a single pocket. If the pocket is not of the buffer type, then it is of constant size. So, we can simply erect vertical walls from the edges and triangulate the resulting cylinders (which is a two-dimensional problem), while also adding enough internal Steiner points to make the facet triangulations compatible (see [1, 8] for a similar idea). If the pocket is a buffer, then we can further assume that it contains no terrain vertex, so there is not even any need to erect walls.

The only difficult case is when a pocket contains one or several drums. The portion of the pocket outside its drums can be handled as before. The question is how to triangulate a superdrum R. Such a polyhedron resembles a thick polygon (Fig. 7). It consists of two facets F, G (arbitrarily complex simple polygons within f and g) connected together by lateral facets. As in a drum, the edges of the lateral facets provide a bijection between the vertices of the polygons F and G. Note that a given pair (f, g) can give rise to more than one polyhedron R.

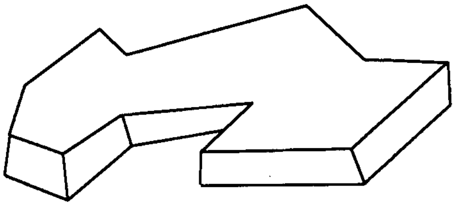


Figure 7.

Naive methods for triangulating R seem to run into all sorts of snags, as one must be careful to keep the size small. Recall that the pockets contributing the drums of R are arranged in a hierarchical fashion. The cross-section of the outer hierarchy of P with the plane supporting F is a nonempty collection of $O(\log n)$ nested convex polygons, C_0, C_1, etc. The $2d$-pockets between them are convex polygons consisting of a base made of some number of collinear segments and a constant-size polygonal arc (Fig. 8). The non-dotted $2d$-pockets in the figure indicate the drums. The non-dotted region, which is precisely the facet F of R, is surrounded by a set of *blocking* $2d$-pockets (the dotted polygons). Note that these blocking pockets might not necessarily be all adjacent to R.

We describe a method for subdividing F and explain how it can be carried over to R. Let B_i be the set of vertices of C_i incident to blocking $2d$-pockets. Note that the B_i are not necessarily disjoint. By analogy, the points of B_i are called *blocking*. We define D_i to be the convex hull of all the points in $B_0 \cup \cdots \cup B_i$.

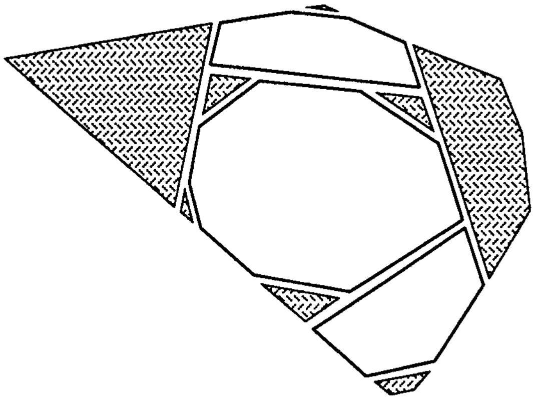


Figure 8.

An easy way to visualize D_i is to think of the blocking points of B_j as nails in a board and C_i as a tight rubber band which is to be snapped: the new position of C_i is precisely D_i.

Recall that we have an identical structure on the facet G, so we can similarly define a polygon D'_i from C'_i. Note, however, that although C_i and C'_i are in bijection, the same need not be true of D_i and D'_i. We now match each D_i and D'_i by taking the convex hull H_i of their union. Since the sequences $D_0, D_1, \ldots$ and $D'_0, D'_1, \ldots$ are nested, the same is true of the sequence of convex hulls H_0, H_1, etc. Let $K_i = (H_i \cap R) \cup H_{i-1}$, so that $R = \cup_{i=1}^{n}(K_i \setminus H_{i-1})$. Note that $K_i \setminus H_{i-1}$ is the region between two nested convex polytopes: it is free of any blocking material, so we can apply Theorem 2.1 to triangulate it.

The Complexity of the Triangulation. An edge of the terrain can appear in only $O(\log n)$ pockets. It follows immediately that the number of tetrahedra produced outside of the drums is $O(n \log n)$. For the same reason the combined complexity of all the merged drums is at most $O(n \log n)$. The only question remaining is how many tetrahedra are produced during the triangulation of the superdrum R. The final triangulation step adds only a constant factor to the size, so it suffices to estimate the total complexity of the H_0, H_1, etc. The size of H_i is proportional to the total number of edges in D_i and D'_i. But notice that the edges in all the D_i's form a planar graph, therefore it suffices to count the number of vertices, which is $|B_0| + |B_1| + \cdots$. Unfortunately, this estimate is too crude. We must take into consideration the fact that not all blocking points contribute to the complexity. In particular, it is clear that only points on blocking $2d$-pockets that are *adjacent* to R can become vertices of the $R \cap H_i$'s. Furthermore, if more than two vertices lie on the base of a blocking $2d$-pocket, only the two endpoints can play any role. It thus appears that the contribution of superdrums to the overall size of the triangulation is at most proportional to the the total number of blocking $2d$-pockets adjacent to superdrums. Such a $2d$-pocket is blocking for one of two reasons:

1. It is intersected by an edge of f or g (Fig. 9).

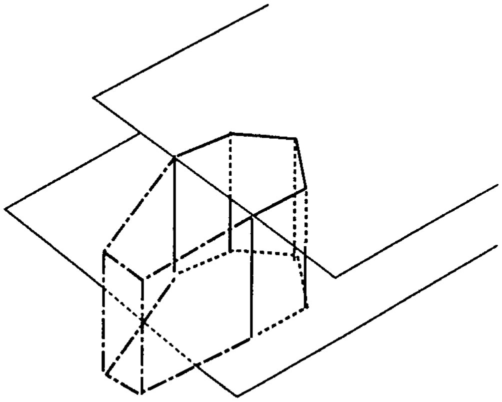


Figure 9.

2. It is not intersected by any edge of f or g but it is adjacent to what would be a drum D, if it were not for a terrain edge that intersects D (Fig. 10).

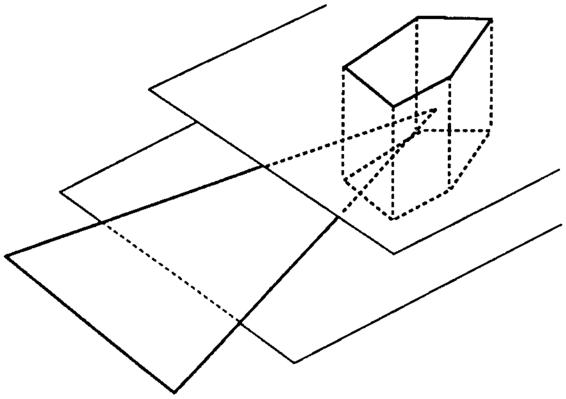


Figure 10.

Obviously, there are at most $O(n \log n)$ $2d$-pockets of the first kind. The same is true of the second kind of pockets, but to see why is a little more subtle. The crux is that no single terrain edge can create more than two $2d$-pockets of type 2 within the same three-dimensional pocket G. The reason for this is that D cuts all across G and therefore no terrain edge can intersect both D and $G \setminus D$. We conclude that the triangulation is of size $O(n \log n)$.

The naive algorithm for constructing the triangulation involves intersecting each facet of the terrain with the entire outer hierarchy, which certainly takes no more than $O(n^2)$ time. Within that running time, it is routine to carry out the entire pruning. It is possible to improve the running time slightly, by using fairly complicated batching techniques based on multidimensional searching. It is not worthwhile pursuing that line of attack. Whether a simple, quasi-linear algorithm exists remains an open question.

3.2 The Lower Bound

We construct a convex polytope and a terrain disjoint from it, such that any convex decomposition of the space between them requires on the order of $n \log n$ pieces, where n is the combined size of the input.

Building the Polyhedra. The polytope is a cylinder with half of a regular $2m$-gon as a base, where $m = 2^p$. It consists of (i) a horizontal square of unit area facing up, (ii) m *slabs* facing down, (iii) two congruent vertical facets whose normals are parallel to the y-axis (Fig. 11).

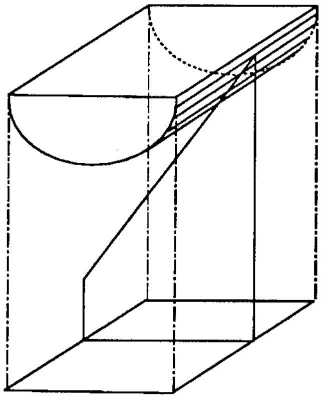


Figure 11.

To define the terrain it is helpful to (conceptually) enclose the cylinder inside a tall box, with the square of the cylinder coinciding with the top of the box. The terrain consists of the plane supporting the bottom of the box, plus m blade-like protrusions abutting the slabs of the cylinder (Fig. 11 shows only one of them). Each protrusion is associated with a slab of the cylinder: it is essentially a trapezoid (with small but positive thickness) parallel to the x-axis; its top side is parallel to a slab of the cylinder and comes extremely close to it without touching it. The trapezoids are in one-to-one correspondence with the m slabs of the cylinder. We assume that each trapezoid extends all across the vertical box. This is why it is necessary to make the box tall enough: a simple calculation shows that the height of the box should be on the order of m. Clearly, the total description size of the input scene is $n = O(m)$.

Our construction is complete once we specify (i) the permutation that defines the correspondence between slabs and trapezoids, and (ii) the placement of the trapezoids along the y-direction. Because the box is tall enough, any permutation on m elements is feasible. We choose the bit-reversal permutation $\pi = (i_0, \ldots, i_{m-1})$, where i_k is 1 plus the number obtained by writing k in binary over p bits and reversing the bits. For example, if $p = 3$, we obtain

$$\pi = (1, 5, 3, 7, 2, 6, 4, 8).$$

Placing Guards. The slabs can be thought of as the leaves of a complete binary tree of depth p. An internal node v thus is naturally associated with a sequence of consecutive slabs $s_i, \ldots, s_j$. Let $W(v)$ be the wedge formed by two planes tangent to the cylinder along the segments $s_{i-1} \cap s_i$ and $s_j \cap s_{j+1}$. We choose the planes so that their normals bisect the angles between their respective slab pairs. If s_i or s_j is an extreme slab then we take the vertical plane supporting the relevant wall of the box. Fig. 12 shows such a wedge (with some liberty in the drawing to make it clearer). Given a node v (not a leaf, not a parent of a leaf, not the root), consider the bounding line L_v of the wedge $W(v)$. A simple geometric observation is that since the trapezoids are extremely close to the slabs, the line L_v intersects (strictly) the trapezoids in correspondence with the slabs $s_i, \ldots, s_j$, and only those. Furthermore, because of the bit-reversal permutation, as we move along the line we encounter trapezoids from the left and right children of v in alternation. The intersection with the trapezoids are such tiny segments that we might as well as think of them as points $q_0, \ldots, q_{j-i}$ (given in ascending y-order). The midpoint of a segment of the form $q_l q_{l+1}$, for $0 \leq l < j - i$, is called a *guard* (the crosses in Fig. 12). Obviously, there is a total of about $m \log m$ guards overall. The lower bound will follow directly from the fact that by a judicious placement of the trapezoids in the y-direction we can ensure that no two guards can see each other.

Instead of assigning each trapezoid an explicit y-coordinate, which would complicate the proof, we define their placement in a recursive manner. We process the tree of trapezoids bottom-up. For a node to be processed means that the entire block of trapezoids associated with it has been placed, up to translation (i.e., its degrees of freedom have been reduced to one). Since we proceed bottom-up, to process v simply means deciding the relative positioning of the blocks associated with the two children of v. To simplify matters further, we make the placements of all the blocks at a given level in the tree identical up to translation. Thus, it suffices to specify the placement of the nodes on the leftmost path of the tree, $v_0, \ldots, v_p$.

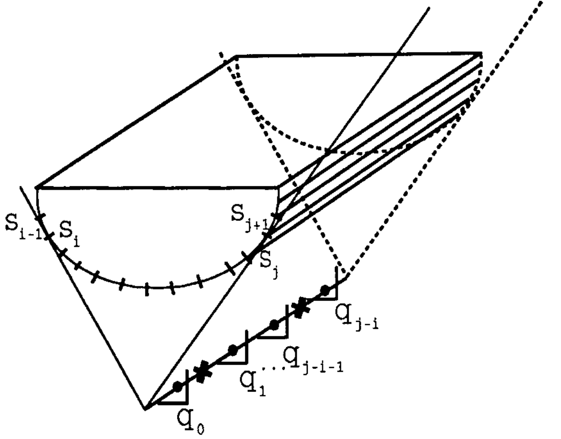


Figure 12.

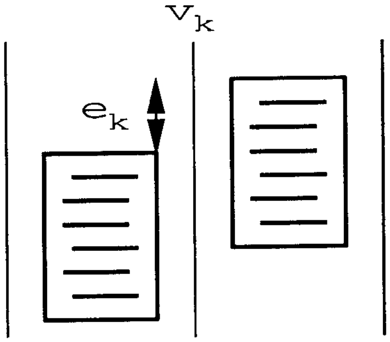


Figure 13.

Let $\varepsilon_k = (c/m)^k$, for some small enough constant $c > 0$. For any $1 \leq k \leq p$, we place the block for the right child of v_k ahead of the block for the left child of v_k by precisely ε_k (Fig. 13). Observe that it is always easy (for c small enough) to fit the entire block of trapezoids within the box. The lower bound follows from the next lemma.

Lemma 3.2 *No two guards can see each other.*

Proof: If the two guards p, q correspond to nodes v, w that are not in any ancestral relationship, then the convex cylinder itself obstructs their mutual visibility. If $v = w$, then the trapezoids passing through the points q_0, q_1, etc., hide the guards from one another. So, the only case remaining is if, say, v is an ancestor of w. Let j and k ($j < k$) be the heights of w and v, respectively. If we project trapezoids and guards onto the xz-plane, we find that not only the projection of p lies in that of every trapezoid associated with v, but actually the projected point is at least at a distance proportional to D/m away from any edge of the trapezoid's projection (Fig. 14), where D is the distance from the bounding line L_v to either of the two lines of contact between the wedge $W(v)$ and the cylinder.

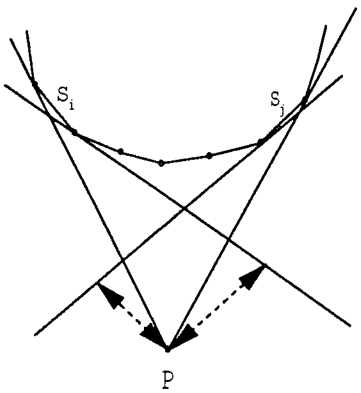


Figure 14.

Thus, there exist two disks of radius $\Omega(D/m)$ which any infinite ray of visibility from p must necessarily avoid: these disks are centered on and normal to the line L_v, and they lie on each side of p at a distance $\varepsilon_k/2$ from it (Fig. 15).

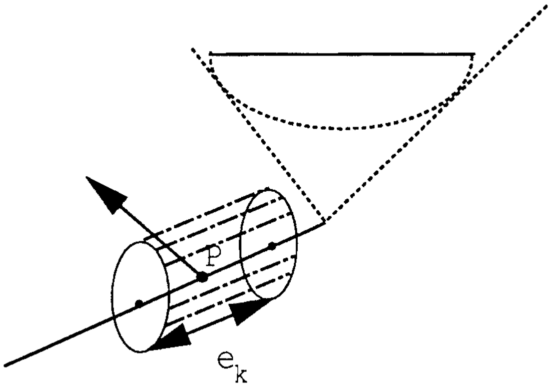


Figure 15.

Let p' be the point of L_w nearest to p (Fig. 16). Because of the two obstructing disks, p cannot see any point of L_w away from p' by more than $O(|pp'|\varepsilon_k m/D)$, which is $O(c|pp'|\varepsilon_j/D)$. But $|pp'|$ is at most D, so by setting c small enough, we find that no point on L_w away from p' by more than $\varepsilon_j/3$ can see p. But q lies at least $(\varepsilon_j - \varepsilon_k)/2$ away from p', which gives a contradiction. □

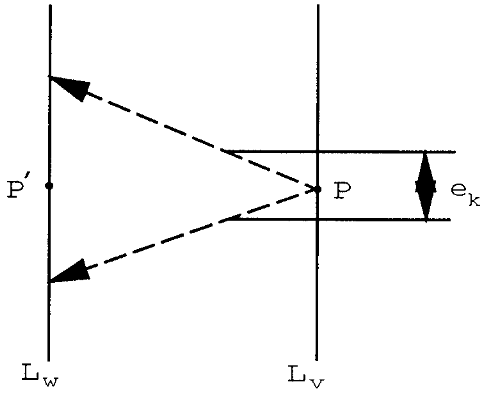


Figure 16.

4 Genus and Reflex Angles

Intuitively, it seems clear that a polyhedron with large genus should have many reflex dihedral angles (we call the corresponding edges *reflex*). Recall that a polyhedron of genus g is a 3-manifold with piecewise-linear boundary that is homeomorphic to a torus with g holes. If r is the number of reflex edges, we will show that

$$g \leq r + 1. \tag{1}$$

The curvature k_v at the vertex v is defined to be

$$k_v = \frac{1}{2\pi}\left(2\pi - \sum_i \theta_i\right),$$

where θ_i is the angle between two consecutive edges incident to v. The Gauss-Bonnet formula relates the total curvature to the genus:

$$\sum_v k_v = 2 - 2g,$$

Inequality (1) follows easily from the next lemma.

Lemma 4.1 *The number of reflex edges incident to a vertex v is at least $-k_v$.*

Proof: Project the facets incident to v onto a unit sphere centered at v. This produces a simple, closed curve made of geodesic pieces. In other words, it is a "polygon" on the sphere made of arcs of great circles. A reflex edge gives rise to a reflex angle on the polygon and vice-versa. It is easy to see that

$$L \leq 2\pi(R + 1), \tag{2}$$

where L is the length of the curve and R is its number of reflex angles. Indeed, if R is zero then the inequality simply states the classical fact that the curvature at a locally convex point is nonnegative. If R is nonzero, then we can draw arcs of great circles from each reflex point along the bisector of its reflex angle, and thus decompose the unit sphere into at most $R+1$ convex regions (i.e., regions where any two points are visible along an arc of great circle connecting them). The case $R=0$ can now be used $R+1$ times to establish (2). The lemma follows immediately from the fact that

$$L = \sum_i \theta_i = 2\pi(1-k_v).$$

□

Chazelle and Palios [8] have given an algorithm for triangulating a genus-0 polyhedron with n vertices and r reflex angles, using $O(n+r^2)$ tetrahedra. The algorithm works for arbitrary genus g, but the bound becomes $O(n+(r+g)^2)$ tetrahedra. The inequality $g \leq r+1$ implies that the true bound is still $O(n+r^2)$. In other words, the complexity bound for genus 0 applies to polyhedra of arbitrary genus as well.

References

[1] Aronov, B., Sharir, M. *Triangles in space or building (and analyzing) castles in the air*, Combinatorica, 10 (1990), 137–173.

[2] Aronov, B., Sharir, M. *The Union of convex polyhedra in three dimensions*, Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci. (1993), 518–527.

[3] Aronov, B., Sharir, M. *Castles in the air revisited*, Proc. 8th Annu. ACM Sympos. Comput. Geom. (1992), 146–156.

[4] Bajaj, C.L., Dey, T.K. *Convex Decompositions of Polyhedra and Robustness, SIAM Journal on Computing* 21 (1992), 339–364.

[5] Bern, M. *Compatible tetrahedralizations*, Proc. 9th Annu. ACM Sympos. Comput. Geom. (1993), 281–288.

[6] Chazelle, B. *Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm*, SIAM Journal on Computing, 13 (1984), 488–507.

[7] Chazelle, B. *An optimal algorithm for intersecting three-dimensional convex polyhedra*, SIAM Journal on Computing, 21 (1992), 671–696.

[8] Chazelle, B., Palios, L. *Triangulating a nonconvex polytope*, Discrete and Computational Geometry, 5 (1990), 505–526.

[9] Dey, T.K. *Triangulation and CSG representation of polyhedra with arbitrary genus*, Proc. 7th Annu. ACM Sympos. Comput. Geom. (1991), 364–372.

[10] Dobkin, D.P., Kirkpatrick, D.G. *Fast detection of polyhedral intersection*, Theoret. Comput. Sci. 27 (1983), 241–253.

[11] Dobkin, D.P., Laszlo, M.J. *Primitives for the manipulation of three-dimensional subdivisions*, Algorithmica, 4 (1989), 3–32.

[12] Martin, A. *A simple primal algorithm for intersecting 3-polyhedra in linear time*, Tech. Report 91–16, July 1991.

[13] Mehlhorn, K. *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry*, Springer-Verlag, Heidelberg, Germany, 1984.

[14] Mulmuley, K. *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice-Hall, 1994.

[15] Pellegrini, M. *Building convex space partitions induced by pairwise interior-disjoint simplices*, ICSI TR–93–039, Aug. 1993.

[16] Ruppert, J., Seidel, R. *On the difficulty of triangulating three-dimensional non-convex polyhedra*, Discrete and Computational Geometry, 7 (1992), 227–253.

[17] Schönhardt, E. *Über die Zerlegung von Dreieckspolyedern in Tetraeder*, Math. Annalen, 98 (1928), 309–312.

An Optimal Bound for Conforming Quality Triangulations
(extended abstract)

Tiow-Seng Tan

Department of Information Systems & Computer Science
National University of Singapore
Lower Kent Ridge, Singapore 0511

Abstract

This paper shows that for any plane geometric graph $\mathcal{G}$ with n vertices, there exists a triangulation $\mathcal{T}$ conforms to $\mathcal{G}$, i.e. each edge of $\mathcal{G}$ is the union of some edges of $\mathcal{T}$, where $\mathcal{T}$ has $O(n^2)$ vertices with angles of its triangles measuring no more than $\frac{11}{15}\pi$. Additionally, $\mathcal{T}$ can be computed in $O(n^2 \log n)$ time. The quadratic bound on the size of its vertex set is within a constant factor of worst case optimal.

1 Introduction

In many engineering applications such as finite element analysis [Cave74], surface interpolation [Laws77], and shape reconstruction [Bois88], two and higher-dimensional domains are frequently decomposed into small and simple elements before numerical computation. One particularly important class of decompositions is the so-called simplicial cell complex, usually referred to as *triangulation*, where the domain is decomposed into simplices—triangles in two and tetrahedra in three dimensions—so that the intersection of two simplices is either empty or a face of both.

For a given domain such as a polygonal region or, more generally, a plane graph with straight edges, there are clearly many ways to decompose or *triangulate* it into a triangulation, with or without the addition of new vertices to the domain. But accuracy and efficiency of an engineering computation impose optimal criteria such as the triangle shape (with bounds on triangle angles away from 0 and π) and vertex size (with bounds on the number of new vertices), respectively. These criteria of shape and size are somewhat conflicting—good triangles may be the result of adding new vertices. Automatic generation of triangulations has been a subject for research since the 1960s. Nevertheless, many interesting results, both practical and theoretical ones, have been discovered in the recent years too; see, for example, [BDE92, BEG90, BeEp91, BMR94, Dey91, MeSo92, Mitc93a, Mitc93b, Mitc94, Rupp93, Tan93] and the survey [BeEp92].

This paper considers triangulating a plane geometric graph (i.e. plane graph with straight edges) using triangles with no large angles. Such resulting triangulations have potential applications in the area of finite element and surface approximation; see, for example, [BaAz76, BaLi84, Greg75]. It shows that triangulating a plane geometric graph of n vertices using angles no larger than $\frac{11}{15}\pi$ requires at most $O(n^2)$ new vertices. A previous result of Mitchell [Mitc93b] achieves angle bound of $\frac{7}{8}\pi$ with $O(n^2 \log n)$ new vertices. This paper thus provides significantly better bounds on triangle shape and vertex size. We also note that the quadratic bound on the vertex size is within a constant factor of worst case optimal. Our construction is conceptually simple though its proof is quite involved. It has some flavor of Edelsbrunner and Tan [EdTa93] and Mitchell [Mitc93b]. The complete proof indeed uses some results derived from the latter.

The paper is organized as follows. Section 2 formalizes the problem. Section 3 provides the outline of the method that proves the quadratic bound, and Sections 4 to 9 discuss its details for angle bound $\alpha = \frac{3}{4}\pi$. Then, Section 10 provides details on implementing the proof and extending it to $\alpha = \frac{11}{15}\pi$, and Section 11 concludes the paper. This extended abstract omits most proofs; the full details can be found in [Tan94].

2 Some Terminology

We first introduce some notions, then define the problem.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

Plane Geometric Graphs. Let S be a set of n *points* or *vertices* in $\Re^2$. An *edge* is a closed line segment connecting two points. Let E be a collection of edges determined by vertices of S. Then $\mathcal{G} = (S, E)$ is a *plane geometric graph* if

(i) no edge contains a vertex other than its endpoints, that is, $ab \cap S = \{a, b\}$ for every edge $ab \in E$, and

(ii) no two edges *cross*, that is, $ab \cap cd \in \{a, b\}$ for every two edges $ab \neq cd$ in E.

One example of a plane geometric graph is a *polygon* where S and E form a single cycle. The cycle is the *boundary* of the polygon. It subdivides $\Re^2$ into a bounded face, its *interior*, and an unbounded face, its *exterior*. A polygon with four edges or sides is a *quadrilateral.*

Triangulations. A *triangulation* is a plane geometric graph $\mathcal{T} = (S, E)$ so that E is maximal. By maximality, edges in E bound the convex hull $ch(S)$ of S (i.e. the smallest convex set in $\Re^2$ that contains S) and subdivide its interior into disjoint faces bounded by triangles. These triangles of $\mathcal{T}$ are referred simply as *triangles*, and their angles as *angles.* With reference to a polygon, we talk about its triangulation as restricted to only its interior.

Conforming Triangulations. A plane geometric graph $\mathcal{T} = (S', E')$ *conforms* to another such graph $\mathcal{G} = (S, E)$ if $S \subseteq S' \subset ch(S)$ and each edge in E is a union of edges in E'. Each vertex in $S' - S$, also termed a *Steiner vertex*, may or may not subdivide an edge of E. A triangulation $\mathcal{T}$ conforms to a plane geometric graph $\mathcal{G}$ is called a *conforming triangulation* of $\mathcal{G}$. The problem studied here is as follows:

> Given a plane geometric graph $\mathcal{G} = (S, E)$, find a conforming triangulation $\mathcal{T}$ of $\mathcal{G}$ with small vertex set and with angles measuring at most α where $\alpha < \pi$.

Previous Results. Two results are directly related to the problem studied here. Bern, Mitchell and Ruppert [BMR94] have recently shown how to augment a polygon (with or without holes) by linear number of Steiner vertices so that it can be triangulated with no obtuse angles. For a general plane geometric graph with n vertices, the recent result of Mitchell [Mitc93b] constructs $O(n^2 \log n)$ Steiner vertices to achieve the angle bound of $\frac{7}{8}\pi$. This bound on the cardinality of Steiner vertices is within a logarithmic factor from the lower bound of $\Omega(n^2)$; see [BeEp91, Mitc93b] for discussions on the lower bound construction (for any angle bounded away from π) due to M. S. Paterson.

3 The Outline of Construction

Given a plane geometric graph $\mathcal{G} = (S, E)$ where $|S| = n$, the algorithm of Edelsbrunner, Tan and Waupotitsch [ETW92] can augment it by edges to a triangulation that minimizes its maximum angle over all possible augmentations. This triangulation is the so-called *min-max angle triangulation* $\mathcal{T}$ of $\mathcal{G}$. So, for a given $\mathcal{G}$, we are done if $\mathcal{T}$ has angles measuring at most α, the targeted angle bound; otherwise, we proceed to *refine* this triangulation by adding Steiner vertices until it has no bad angle. We say an angle is *bad* if it has measure larger than α; otherwise it is *good.* A *good triangle* is a triangle with only good angles whereas a *bad triangle* has some bad angle. A polygon (or polygonal region) that can be triangulated with no bad triangles is said to have a *good triangulation.* For our discussion till Section 10, α is set to $\frac{3}{4}\pi$.

Let us assume now that $\mathcal{T}$ has some bad angles. Suppose $\triangle pqr$ is one with a bad angle at q. Then, in refining $\mathcal{T}$, we need to find a vertex t so that qt subdivides the angle at q into two smaller ones. Notice that such a t may not be any of the existing vertices of $\mathcal{T}$. So, t may possibly be a Steiner vertex. Also, using only one vertex t inside $\triangle pqr$ does not improve the situation since $\angle ptr > \angle pqr$. In general, t on pr is chosen to subdivide the angle at q as well as $\triangle pqr$ into two new triangles. Such a refinement results in the other triangle sharing pr, say $\triangle prs$, to have a vertex on pr. So a subdivision on $\triangle prs$ is now necessary to get rid of the "large angle" of π at t. This situation may continue to *propagate* to other triangles. The main idea of our construction is to bound such *propagations* of vertices.

The proposed construction has six major phases. The first phase (Section 4) subdivides each triangle of $\mathcal{T}$ into three quadrilaterals by adding edges, termed *spokes*, perpendicular to its three sides from the center of its inscribing circle. Some of these quadrilaterals are marked as *fences*; they together with *traps* set up in the second phase (Section 5) help in controlling propagations of vertices. The third phase (Section 6) propagates vertices of spokes and traps. From one vertex, we may get a sequence of vertices where one propagates to the next. This sequence can be viewed as a kind of path, called *propagation path*, whose vertices are possibly Steiner vertices. The fourth phase (Section 7) removes redundant subpaths and the fifth phase (Section 8) untangles crossings of paths. The remaining vertices of paths form the set of required Steiner vertices, and the final phase (Section 9) triangulates each quadrilateral having Steiner vertices on its boundary using edges of paths as guides.

We note that the construction above as it is may add unnecessarily many Steiner vertices. This can nevertheless be avoided by modifying some steps as described in Section 10. Let us next define a few more notions before discussing the details of the construction.

Fences and Traps. A *fence* is a polygon that has a good triangulation even if its edges contain Steiner vertices. A *trap* is a polygon with a *base* edge ab and *dead ends* a', b' where a and b are connected to the respective dead ends via paths $P_{aa'}$ and $P_{bb'}$. Other than a' and b' may be the same point, $P_{aa'}$ and $P_{bb'}$, also termed *the boundary paths of the trap*, do not intersect. Additionally, $P_{aa'}$ with vertices $a = p_0, p_1, p_2, \ldots, p_k = a'$ and $P_{bb'}$ with vertices $b = q_0, q_1, q_2, \ldots, q_k = b'$ are such that $p_{i-1}p_i$ and $q_{i-1}q_i$, for $1 \leq i \leq k$, lie in the closure of some quadrilateral. This condition implies that a trap encloses neither vertices of S nor endpoints of spokes. Vertices of a trap are *not* necessarily Steiner vertices in the final construction.

The construction of fences and traps in Sections 4 and 5 considers the following issues:

- building linear number of fences and traps, and
- ensuring each trap has at most linear number of vertices.

Paths. A *path* with *origin* p_0 is a sequence of vertices $p_0, p_1, \ldots, p_i, p_{i+1}, \ldots$ where p_i and p_{i+1} are points on two different edges of some quadrilateral. Two consecutive vertices p_i, p_{i+1} of the path define an *edge* p_ip_{i+1} or, more specifically, a *directed edge* $\vec{p_i}p_{i+1}$ with *tail* p_i and *head* p_{i+1}. Each directed edge defines four angles at its two endpoints with edges of the quadrilateral containing it. For our purposes, we use a *path* to refer to one where the two angles at the tail of each directed edge are good, and analogously an *inverse path* to one where the two angles at the head of each directed edge are good. And, vertices of paths and inverse paths are always some points on edges of $\mathcal{T}$.

The *length* of a path or an inverse path is its number of vertices minus one, and the *direction* is the direction of its directed edges. Two paths are *coherent* if they traverse through the same set of quadrilateral edges, and are *parallel* if they are coherent and their two edges in the same quadrilateral are parallel. We also use coherent and parallel to describe two inverse paths, or one path with an inverse path.

Two directed edges $\vec{ab}$ and $\vec{cd}$ (of some paths) in quadrilateral $pq'cr'$ are *vertex disjoint* if $\{a,b\} \cap \{c,d\} = \emptyset$, and *cross* if they are vertex disjoint and $ab \cap cd \neq \emptyset$. They are said to be of the *same orientation* if a and c are on the same edge of $pq'cr'$ (so are b and d), otherwise *opposite orientation*.

Major issues regarding propagation paths in Section 6 are:

- tracing linear number of paths, and
- ensuring each path has length at most linear.

Pebbles. Propagation paths leave traces of its vertices on boundaries of quadrilaterals. A quadrilateral with all traces on its boundary taken as Steiner vertices may not have a good triangulation for traces that are too near to each other (unless the quadrilateral is a fence). Thus, portions of some propagation paths need to be removed and the remaining portions reconnected to nearby ones in the same direction. The set of propagation paths remained plus the set of centers of inscribing circles form the required set of Steiner vertices (or pebbles) for $\mathcal{T}$.

Two major concerns of Sections 7 to 9 are:

- ensuring paths terminate on convex hull edges, fence edges, endpoints of spokes or dead ends, and
- triangulating quadrilaterals whose boundaries contain Steiner vertices using no bad triangles.

Analyzing Cost. Once all the above issues are resolved, the number of Steiner vertices used is clearly quadratic in the number of vertices of $\mathcal{T}$.

4 Erecting Fences

The purpose of a fence is to block or terminate propagations of vertices. We can consider convex hull edges of $\mathcal{T} = (S, E)$ as degenerate fences since no propagation of vertices out of the convex hull is necessary. They are, however, insufficient to block all propagations; we need the following type of fences plus traps discussed in the next section. Throughout the discussion, we refer to some specific measures of angle. As mentioned, $\alpha = \frac{3}{4}\pi$ is the targeted angle bound. Let $\phi = \frac{\pi}{2}$ and $\beta = \frac{\pi}{4}$. Notice that $\alpha = \pi - \beta$ and $\phi + 2\beta = \pi$. We use the term *isosceles triangle* to mean a triangle with apex angle ϕ and two base angles β.

Consider a triangle $pqr \in \mathcal{T}$. Let c be the center of its inscribing circle, i.e. c is the point common to the three angle bisectors. Let p', q', r' be the three perpendicular projections of c onto respectively qr, rp, pq; see Figure 4.1. Edges cp', cq' and cr' are *spokes* of $\triangle pqr$.

Step 1. Erecting Fences.
For each $\triangle pqr \in \mathcal{T}$ with the largest angle at say q, add the center c of its inscribing circle as a Steiner vertex. Also, add spokes cr', cp' and cq' to subdivide $\triangle pqr$ into three quadrilaterals $pq'cr', rq'cp'$, and $qr'cp'$. Then, mark $qr'cp'$ as a fence. And, mark $pq'cr'$ if $\angle p \geq \beta$, and $rq'cp'$ if $\angle r \geq \beta$.

The above step marks quadrilaterals that indeed satisfy the definition of a fence as long as no Steiner vertices

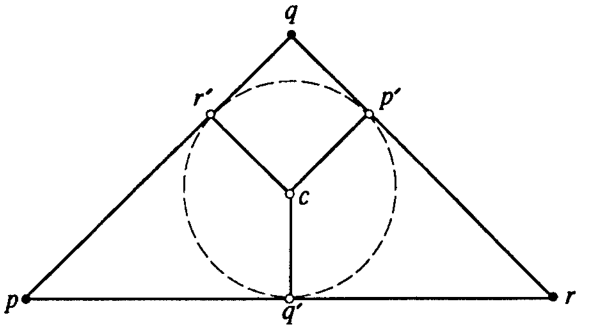


Figure 4.1: Triangle pqr is subdivided into three quadrilaterals $pq'cr'$, $rq'cp'$ and $qr'cp'$. It is simple to check that $|pr'| = |pq'|$, $|rp'| = |rq'|$, $|qr'| = |qp'|$, and $|cr'| = |cp'| = |cq'| = \rho$ where ρ is the radius of the inscribing circle of Δpqr.

will be added to spokes of triangles. This follows immediately from the next lemma. Those quadrilaterals that are not marked as fences are referred as *non-fences.*

Lemma 4.1 Suppose quadrilateral $pq'cr'$ is such that $\angle q' = \angle r' = \frac{\pi}{2}$ and $\angle p \geq \beta$. Then it has a good triangulation even if pr' and pq' contain some Steiner vertices. □

5 Setting Traps

To supplement fences that block some propagations, we step up traps whose dead ends can serve to terminate the other propagations. To do this, we first identify portions of quadrilateral edges, other than spokes, that must be bases of traps. Clearly, we need only consider those that are not parts of the boundaries of fences. We thus talk about quadrilateral $pq'cr'$ in the following as one with right angles at r' and q' and $\angle p < \beta$, unless stated otherwise. Also, we assume $pq'cr'$ lies inside Δpqr of $\mathcal{T}$.

To plan for traps, we introduce a special type of inverse path, termed *i-path*, where all its directed edges define good angles of β at their heads and all such good angles are on the same side of the inverse path. We refer to vertices of i-paths as *i-vertices*. These vertices may not be Steiner vertices in the final construction. In the next step, each i-path constructed has length bounded by K where K is one more than four times the number of edges of $\mathcal{T}$.

Step 2. Planning Traps.

Consider each quadrilateral (non-fence) $pq'cr'$; see Figure 5.1. Trace an i-path at r' with its second vertex r'' on pq' so that $\angle r'r''q' = \beta$. Similarly, trace an i-path at q' with its second vertex q'' on pr' so that $\angle q'q''r' = \beta$. Suppose r_1 and q_2 are vertices on pq and pr, respectively of the two quadrilaterals adjacent to $pq'cr'$ and incident to p. If q_2 lies on pr'', trace out two i-paths at q_2 such that the first edges of the i-paths meet pr' at two points that form an isosceles triangle with apex at q_2. Similarly, trace out two i-paths at r_1 if r_1 lies on pq''. In all the above, we terminate an i-path when it hits either a convex hull edge, a fence edge, a spoke, or when its length has reached K.

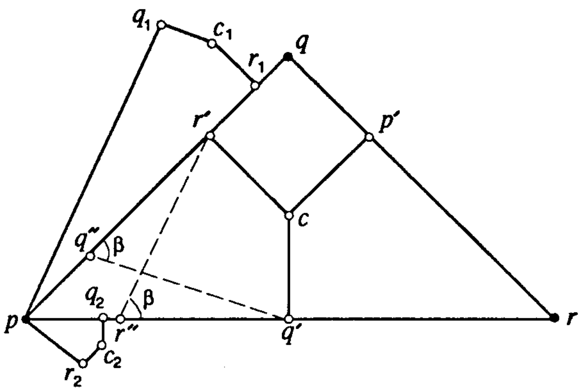


Figure 5.1: For $pq'cr'$, one i-path each is traced at r' and q', and two at q_2 but none at r_1.

The above step traces out a maximum of four i-paths at each endpoint of a spoke—one each into the quadrilaterals (non-fences) where the endpoint is a vertex, and two into the other quadrilateral with the endpoint on one of its edges. Notice that the latter two i-paths at an endpoint may not be coherent, though they started off by moving to the same edge. With all the i-vertices in place, we can identify (next two paragraphs) some portions of quadrilateral edges as bases of traps to be constructed.

Consider a directed edge $p_i\vec{p}_{i+1}$ of an i-path inside a quadrilateral $pq'cr'$ with $p_i \in pr'$ and $p_{i+1} \in pq'$. The head p_{i+1} subdivides pq' into two segments: one that form an angle of β with p_ip_{i+1} and the other of $\pi - \beta$. The former is a *positive segment* and the latter (excluding point p_{i+1}) a *negative segment.* The collection of heads of directed edges of i-paths (inside $pq'cr'$) on pq' subdivide points on pq' into two types: a *positive point* is one in at least a positive segment, and a *negative point* otherwise. Points p and q' are (arbitrarily) said to be positive. By the formulation, pq' is subdivided into three (possibly degenerate) portions by *markers* say m_1 and m_2 where $m_1 \in pm_2$. And, the two portions pm_1 and m_2q' are *I-types* containing only positive points, and m_1m_2 (excluding points m_1 and m_2) is *O-type* containing only negative points.

Notice that the above description of an edge is with reference to a quadrilateral incident to it. So, an edge is subdivided into O-types and I-types on its two sides due to at most three incident quadrilaterals. So there are at most three portions of O-types. These are possible bases of traps to be constructed. Let us assume without loss of generality that $\mathcal{T}$ contains no horizontal edges.

Step 3. Setting Traps.
For each vertex $p \in S$, consider the at most two quadrilaterals that overlap an arbitrarily small horizontal line segment having p as an interior point. Pick from each one edge say pq' as a *candidate edge* to construct traps: for each quadrilateral incident to pq', construct a trap with base $xy \subseteq pq'$ into the quadrilateral where xy is O-type.

The above construction of a trap with base xy on pq' in the direction into $p_1q_1'c_1r_1'$ is as follows. We need the notion of a t-path (the reverse of an i-path). A *t-path* is a path where all its directed edges define good angles of β at their tails and all such good angles are on the same side of the path.

> Trace two (coherent) t-paths at x and y into $p_1q_1'c_1r_1'$ so that those good angles of β at tails of directed edges are on the same side with respect to each t-path. Both t-paths continue until they have just reached say x_l (possibly x) and y_l (possibly y), on p_lq_l' of quadrilateral $p_lq_l'c_lr_l'$ where one of the following is true:
>
> 1. the t-paths intersect at a point, say z_l, in the interior of $p_lq_l'c_lr_l'$ when continue to be extended;
> 2. $p_lq_l'c_lr_l'$ is a fence or p_lq_l' is a convex hull edge; or
> 3. p_lr_l is the very first edge visited for the fifth times by the t-paths where $\triangle p_lr_lq_l \in \mathcal{T}$ contains $p_lq_l'c_lr_l'$.
>
> For cases 2 and 3, x_l and y_l are the dead ends and both t-paths are the boundary paths of the trap. For case 1, consider $z \in p_lr_l'$ where zz_l is perpendicular to x_ly_l. We take z as the dead end if zz_l does not intersect c_lr_l'; otherwise, take r_l' as the dead end. And, we complete the boundary paths of the trap by extending both t-paths till the dead end.

Though case 1 modifies t-paths slightly in constructing traps, it is easy to check that each trap does have its base connected to its dead end(s) by two paths; see also the next lemma.

Lemma 5.1 In the construction of a trap with base xy, the two t-paths from x and y are coherent. □

Lemma 5.2 The number of traps constructed is at most $6n$. □

Lemma 5.3 Each trap has at most K edges. □

6 Propagating Paths

The necessity to propagate vertices to compute quality triangulation is a consequence of the following simple lemma, which was introduced in Edelsbrunner, Tan and Waupotitsch [ETW92].

Lemma 6.1 If xy is an edge in a triangulation $\mathcal{T}$ of a point set S, then $\mu(\mathcal{T}) \geq \max_{s\in S} \angle xsy$ where $\mu(\mathcal{T})$ denotes the largest angle in $\mathcal{T}$. □

In effect, propagation is to subdivide each long edge into small ones so to remove large angles extended by the edge.

To trace a path, we adapt a few terminology introduced in [Mitc93b]. Consider a quadrilateral $pq'cr'$ with $\angle p < \beta$ and right angles at q' and r'. Let s be a point on pq'. The *cone* at s consists of all points t of $pq'cr'$ such that both $\angle tsp$ and $\angle tsq'$ are good (i.e. at most α). Next, the *maw* of s is the portion of the cone at s on the boundary of $pq'cr'$, except for s. Notice that each point on the maw of s is a candidate for extending a path from s. Next, the iterative construction of union of cones at points in maws, starting with s, results in the so-called *horn* of s. Initially at stage 0, the horn is the cone at s. The horn at stage $i+1$ is the union of cones at points in the union of maws, or simply maw, at stage i. It is clear that the two paths bounding the horn of s are actually t-paths starting at s. They are referred as the *boundary paths of the horn.* The *center path* of the horn is the sequence of line segments where each connects the midpoints of the maws (which are *center path points*) for two consecutive stages, starting at s. (We can assume because of Lemma 6.2 below that each maw contains only points on one edge of a quadrilateral, so the midpoint of the maw at a stage is well defined.)

Given some point t in the maw of s at stage j, it is trivial to work from t back to s to obtain a path of length $j+1$ with origin s and last vertex t. This is how the propagation phase constructs a propagation path for a vertex s by generating the horn of s iteratively until it intersects (or covers) some point t that is an endpoint of a spoke, a dead end, or a point on a fence or a convex hull edge. Besides these, t may possibly be some point common to cones of the horn from different stages—which happens when the horn is *self-intersecting* inside some quadrilateral. In this case, the chosen t has to appear as the last vertex as well as some intermediate vertex of the constructed propagation path. One such possible t is a point in the current maw of stage j that is also a center path point of stage i for $i < j$ and the horn defined from t contains t. A propagation path constructed using this condition has length $j + 1$ and consists of a subpath from s to t followed by a loop containing t as derived from the horn of t.

Due to a technical reason (Section 7), we need to distinguish two types of self-intersecting horns—one intersects while passing through a quadrilateral for the second time in the same orientation to one intersects while passing through in opposite orientation. It is the latter that we do not use for constructing propagation path. This is to avoid terminating a propagation path with directed edge $q_i\vec{r}_j$ such that the last vertex r_j (other than an origin of some propagation path) being also a tail of other directed edge inside the quadrilateral containing $q_i\vec{r}_j$. (On the other hand, it is possible that a vertex, other than dead ends and endpoints of spokes, is *passed through* by two different propagation paths in opposite directions. This does not create technical difficulties for Section 7. But, for convenience, we will interpret such a vertex as two nearby vertices in Section 7.)

We say a propagation path P *terminates properly* if its last vertex r_j is an endpoint of a spoke, a dead end, a point on a fence or a convex hull edge, or a head common to another directed edge of P or other propagation path inside the quadrilateral containing the last directed edge of P.

Step 4. Propagating Paths.
Take endpoints of spokes and dead ends of traps as the current Steiner vertices of $\mathcal{T}$. These Steiner vertices, except for centers of inscribing circles, are origins of propagation paths to be generated. For an endpoint of a spoke on an edge of $\mathcal{T}$, we generate a propagation path in the direction out of the triangle containing the spoke. For a dead end (which is not also an endpoint of a spoke), we generate two distinct propagation paths—one each into the two quadrilaterals containing the dead end on their boundaries. All propagation paths are to terminate properly.

The following two addenda to Step 4 are to ensure that the lengths of all propagation paths are bounded by $O(n)$. Firstly, Step 4a is performed to generate some new origins of propagation paths before the normal processing of Step 4. This extra effort is to avoid using horns self-intersect in opposite orientations to create propagation paths. Secondly, Step 4b handles an exceptional case due to Step 4 so to cut short some propagation paths.

Step 4a. Propagating Paths—Beginning Case.
For each endpoint s of a spoke that lies on some edge of $\mathcal{T}$, we consider the two t-paths with origin s in the direction of a propagation path to be constructed for s (in Step 4). These t-paths sandwich an angle of $\pi - 2\beta$ at s and each terminates once (1) it visits an edge of $\mathcal{T}$ twice or (2) it crosses some spoke or arrives at a fence or a convex hull edge. Only for (1), the last vertex of the t-path becomes an origin as if it is a dead end which Step 4 uses to generate two propagation paths.

Step 4b. Propagating Paths—Exceptional Case.
Suppose in Step 4, a propagation path remains undetermined for s after its horn has since entered the base xy of a trap, say T, and traversed the same number of edges as T to reach the edge containing the two dead ends, x_d and y_d, of T. Let w be any point in the intersection of the horn of s with xy, and let a_w and b_w be the two points of the boundary paths of the horn of w (in T) on x_dy_d. Treat a_w and b_w as two additional dead ends just like the other dead ends identified earlier on. So each is an origin of two propagation paths to be generated by Step 4, one into T through x_dy_d and one out of T. And, we complete the construction for s by a propagation path that terminates at a_w or b_w.

A few remarks are in order. First, the two t-paths of s in Step 4a actually bounds the horn at s initially, i.e. they are coherent. But they are not necessarily coherent in extending new directed edges and one may actually be longer than the other eventually. Note that we talk of these t-paths without requiring their vertices be Steiner vertices in the construction so far. Second, when both t-paths end in case (1), we can actually create just one new dead end instead of two as stated. For this, we then fix one t-path as the initial part of the propagation path to be constructed for s, and apply Step 4 to the last vertex of the t-path to complete the remaining part of the propagation path of s. This way avoids creating two other propagation paths. Third, we will see that the above special arrangement of Step 4b is needed at most once for each trap (with two dead ends). So we generate only $O(n)$ propagation paths in total. Fourth, the horn of w in T is properly contained in T, as the horn of s does not meet the boundary paths of T; otherwise, we would have a propagation path with origin s that terminates properly at one dead end of T. So, a_w and b_w are well defined. Again, we talk of the horn of w without requiring vertices on its boundary paths be Steiner vertices.

The next result implies that once a horn intersects some spokes, we can construct from the horn a propagation path that terminates properly. So, we avoid propagation paths that cross spokes.

Lemma 6.2 Let s be a point on pq' of quadrilateral $pq'cr'$. If the maw of s contains points on cq' or cr', then the maw of s contains r'. □

Lemma 6.3 For a propagation path P to return to an edge of $\mathcal{T}$, P has to visit some candidate edge (identified in Step 3). □

Lemma 6.4 If the horn of s self-intersects in opposite orientation at stage ℓ, then there exists a propagation path with origin s that terminates properly and has less than $\ell + 3n$ vertices. □

Lemma 6.5 Each propagation path constructed by Step 4 has less than $15n$ vertices. □

Lemma 6.6 Step 4 computes less than $66n$ propagation paths. □

7 Removing Complications

The result of the propagation phase (Step 4) is another plane geometric graph conforms to $\mathcal{T}$. Each vertex s on an edge, say pq', of a quadrilateral (non-fence) $pq'cr'$ is such that there is another vertex t on pr' where st is an edge of some propagation path passing through $pq'cr'$. So, either the two angles at s or that at t inside $pq'cr'$ are good. Because of this, quadrilaterals with vertices on their boundaries can be triangulated with no bad triangles except when there are vertices due to nearby directed edges of the same orientation. As such, we use the following merging step to remove this "complication". Vertices of $pq'cr'$ along pq' are labeled as $p = q_0, q_1, q_2, \ldots, q_l = q'$ and along pr' as $p = r_0, r_1, r_2, \ldots, r_m = r'$. A *neighboring pair of directed edges* are two directed edges of the same orientation such that the segment defined by their tails contains no other tail.

Step 5. Merging Paths.
For each quadrilateral $pq'cr'$, repeat the following so long there exists a neighboring pair $q_i\vec{r}_{i'}$ and $q_j\vec{r}_{j'}$, with $i < j$ and good $\angle r_{i'}q_jp$ and $\angle r_{i'}q_jq'$ or, analogously, a neighboring pair $r_i\vec{q}_{i'}$ and $r_j\vec{q}_{j'}$ with $i < j$ and good $\angle q_{i'}r_jp$ and $\angle q_{i'}r_jr'$. We just describe the former case. Let P_s be the shortest subpath of some propagation path P starting with $q_j\vec{r}_{j'}$ until a vertex of P that is a head common to two (or more) directed edges, or a vertex on a fence or convex hull edge, or an origin of other propagation paths. We remove edges and vertices of P_s except possibly for the last vertex of P_s that is shared by other propagation paths, and terminate P by new edge $q_j\vec{r}_{i'}$.

Vertices of spokes and of propagation paths remained after Step 5 form the final set of Steiner vertices for $\mathcal{T}$.

Lemma 7.1 All propagation paths still terminate properly with each merging of paths. □

Lemma 7.2 Once Step 5 has completed, any two directed edges of the same orientation inside a quadrilateral do not cross. □

Lemma 7.3 Once Step 5 has completed, any two directed edges that are vertex disjoint and of the same orientation inside a quadrilateral define a region that has a good triangulation. □

8 Untangling Crossings

For this and the next section, we view each directed edge as an individual with two good angles at its tail, without associating it to a particular propagation path. Though Step 5 (Section 7) also removes some crossings, directed edges of opposite orientations may still cross inside a quadrilateral. With reference to a quadrilateral (non-fence) $pq'cr'$, the next step is to get rid of all crossings by modifying edges while maintaining the following invariants:

I-1. each edge is assigned an orientation with its tail defining two good angles;

I-2. each Steiner vertex is an endpoint of some directed edge;

I-3. no Steiner vertex is a tail of more than one directed edge;

I-4. directed edges of the same orientation do not cross;

I-5. for two (non-crossing) vertex disjoint directed edges $q_i\vec{r}_{i'}$ and $q_j\vec{r}_{j'}$ with $i < j$, the region defined by these four vertices can be subdivided into two good triangles using $r_{i'}q_j$. Analogous assertion applies to two vertex disjoint directed edges $r_{i'}\vec{q}_i$ and $r_{j'}\vec{q}_j$.

These invariants are true initially. In particular, I-1, I-2 and I-3 are trivially true, and I-4 and I-5 are true due to Lemma 7.2 and Lemma 7.3 respectively. In Step 6, the *next available index from i* refers to the smallest index, starting at i and in increasing order, whose corresponding vertex can be used as the head of a new directed edge while maintaining I-4. We say an ordered pair

(a, b) is lexicographically smaller than another ordered pair (a', b') if $a < a'$, or $a = a'$ and $b < b'$.

Step 6. Removing Crossings

Consider each quadrilateral $pq'cr'$ in turn to remove crossings. The following two cases are repeated until all crossings are removed. Each iteration involves edges $q_i r_{i_1}$ crosses $r_j q_{j_1}$ such that $(i + j, i_1 + j_1)$ is lexicographically the smallest ordered pair. See Figure 8.1.

Case A. $q_i\vec{r}_{i_1}$ crosses $r_j\vec{q}_{j_1}$.

We first add $q_i r_j$ with the appropriate orientation and remove $q_i\vec{r}_{i_1}$ and $r_j\vec{q}_{j_1}$. Next, there are two symmetrical cases depending on whether $q_{j_1}r_{i_1}$ defines two good angles at q_{j_1} or at r_{i_1}. Let us discuss only the former. If q_{j_1} is no longer a vertex of any directed edge, we add $q_{j_1}\vec{r}_{i_2}$ where i_2 is the next available index from i_1. Next, if r_{i_1} is no longer a vertex of any directed edge, we add $r_{i_1}\vec{q}_{j_2}$ where j_2 is the next available index from j_1.

Case B. $r_{i_1}\vec{q}_i$ crosses $q_{j_1}\vec{r}_j$.

We add $q_i\vec{r}_j$ if q_i is not a tail of any directed edge, else add $r_j\vec{q}_i$ if r_j is not a tail of any directed edge. Next, if r_{i_1} (q_{j_1}, respectively) is a vertex of other directed edges, we are done by removing $r_{i_1}\vec{q}_i$ ($q_{j_1}\vec{r}_j$, respectively). Otherwise, move all directed edges (inclusive of $r_{i_1}\vec{q}_i$) crossing $q_{j_1}\vec{r}_j$ with heads at q_i to $q_{j'}$ where $j' = i + 1$.

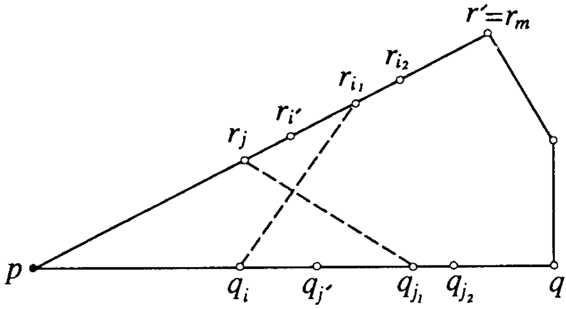


Figure 8.1: Since directed edges of the same orientation do not cross (invariant I-4), the underlying directed edges based on the two chosen line segments $q_i r_{i_1}$ and $r_j q_{j_1}$ must have opposite orientations. Any directed edge e with endpoint on $r_{i'}$ for $j \leq i' < i_1$ must cross $q_i r_{i_1}$ by the choice of $(i+j, i_1+j_1)$. As such and because of I-4, the other endpoint of e must be q_{j_2} for $j_2 \geq j_1$ with strict inequality for Case B, and e must have the same orientation as the directed edge based on $r_j q_{j_1}$. A similar statement applies to directed edges with endpoints on $q_{j'}$ for $i \leq j' < j_1$.

It is clear that each iteration of either Case A or Case B may introduce new crossings. Still, the process can be shown to terminate.

Lemma 8.1 When applied to a quadrilateral $pq'cr'$, Step 6 terminates by removing all crossings. □

Lemma 8.2 Invariants I-1 to I-5 are maintained after each iteration of Case A. □

Lemma 8.3 Invariants I-1 to I-5 are maintained after each iteration of Case B. □

9 Triangulating Quadrilaterals

We now have for each quadrilateral $pq'cr'$ some Steiner vertices on its boundary and directed edges in its interior. Each Steiner vertex lies on some directed edge, and each directed edge defines two good angles at its tail. Since directed edges do not cross, they subdivide $pq'cr'$ into regions of three, four, or five sides. We label these regions in their natural sequence starting with that containing c as R_1 then its adjacent one as R_2 and so on until that containing p as $R_{m'}$. All except possibly for R_1 have straightforward good triangulations. We must however first handle R_1 by merging nearby regions to R_1 until we have obtained a region with a good triangulation. As such, some directed edges may not be in the final triangulation—they serve as guides in our construction.

Step 7. Triangulating Quadrilaterals.

Quadrilaterals with no Steiner vertices or are fences can easily be triangulated with no bad triangles; refer to Lemma 4.1. We next consider each non-fence $pq'cr'$. Label its regions subdivided by directed edges from that containing c to that containing p as $R_1, R_2, \ldots, R_{m'}$. Starting with $\mathcal{R} = \emptyset$ and $k = 0$, we then repeat incrementing k and including R_k into $\mathcal{R}$ until we can produce a good triangulation for $\mathcal{R} = \cup_{i=1}^{k} R_i$. Then for the remaining regions $R_j, j = k+1, k+2 \ldots, m'$ of three or four sides, we triangulate each in a straightforward way.

Lemma 9.1 Each region R_k for $k = 2, 3, \ldots, m'$ has a good triangulation. □

Lemma 9.2 There exists a $k \leq m'$ such that $\mathcal{R} = \cup_{i=1}^{k} R_i$ has a good triangulation. □

Till here, we have the following theorem:

Theorem 9.3 Triangulating a plane geometric graph $\mathcal{G} = (S, E)$ of $|S| = n$ vertices and $|E| = O(n)$ edges using angles no larger than $\frac{3}{4}\pi$ requires $O(n^2)$ Steiner vertices. □

10 Implementing Construction

In this section, we describe an efficient algorithm to implement the above constructive proof. Also, we discuss ways to avoid some redundant Steiner vertices, and extend the construction to a better angle bound of $\alpha = \frac{11}{15}\pi$. Let us assume that each point coordinate can be stored in a constant amount of storage and that basic geometric operations, such as projecting a point onto a line can be carried out in a constant amount of time.

Theorem 10.1 Triangulating a plane geometric graph $\mathcal{G} = (S, E)$ of $|S| = n$ vertices and $|E| = O(n)$ edges using angles no larger than $\frac{3}{4}\pi$ requires $O(n^2)$ storage and $O(n^2 \log n)$ time. □

Reducing Steiner Vertices. The above construction generates substantially many Steiner vertices, though it achieves the worst case optimal bound. As implied by Lemmas 6.5 and 6.6, the constant factor in the quadratic bound is relatively large. Fortunately, we see in the following a modified construction that avoids many redundant propagation paths and Steiner vertices. Let us start with Step 1. Instead of subdividing each triangle pqr of $\mathcal{T}$ with largest angle at q into three quadrilaterals, we subdivide it into two or three triangles as follows depending on $\angle q$.

Case a. $\angle q > \alpha$.
Subdivide $\triangle pqr$ into $\triangle pqq'$ and $\triangle rqq'$ where $q' \in pr$ and $\angle pqq' = \angle rqq'$. We treat qq' as a spoke.

Case b. $2\beta \leq \angle q \leq \alpha$.
Same as Case a. Additionally, mark $\triangle pqq'$ as a fence if $\angle p \geq \beta$, and similarly mark $\triangle rqq'$ if $\angle r \geq \beta$.

Case c. $\angle q < 2\beta$.
Subdivide $\triangle pqr$ into $\triangle pr'q'$, $\triangle qp'r'$, $\triangle rq'p'$ and $\triangle p'q'r'$ where $p' \in qr, q' \in pr, r' \in pq$ are so that $pq' = pr', qp' = qr', rq' = rp'$. Also, treat edges of $\triangle p'q'r'$ as spokes, and mark each of the other three triangles as a fence if its angle opposite its edge incident to $\triangle p'q'r'$ is no less than β.

With this subdivision, we then plan and set up traps in Step 2 and Step 3, and create other dead ends in Step 4a. As before, we next generate propagation paths in Step 4 (and Step 4b) but only for selected vertices. Each q' obtained in Case a is an origin of a propagation path out of the triangle containing the spoke with q' as an endpoint. We next consider whether more propagation paths are to be created for s where s is either q' in Case b, p' or q' or r' in Case c, or a dead end. When s terminates some propagation path P' generated so far, we generate a propagation path P'' for it following P' (so we can imagine P' to include also the part of P''); otherwise, it is ignored in Step 4 and is not considered a Steiner vertex in later steps.

Step 5 and Step 6 are the same as before. The former actually further removes away some unnecessary Steiner vertices. Note that a dead end can be treated as an ordinary vertex subject to removal by Step 5 if the dead end originates one instead of two propagation paths (in opposite directions) at the completion of Step 4. A similar consideration applies to an endpoint of a spoke. Finally, Step 7 works quite the same as before: for a non-fence, we just use the straightforward method in Lemma 9.1; otherwise, we have a fence with $\beta \leq \angle p < 2\beta$ that can be solved with Lemma 4.1. However, there is one problem—let us pick Case c for the discussion, and note that a similar situation occurs for Case b. Vertices p', q', r' used in Step 1 to subdivide $\triangle pqr$ into three triangles are not necessarily Steiner vertices as mentioned. So the resulting triangulation obtained from Lemma 9.1 or Lemma 4.1 may have triangles with vertices that are not actually created. These triangles have to be discarded. As a result, we obtain an untriangulated region R supported by edges of $\triangle pqr$ and directed edges inside $\triangle pqr$. It can be proved that R resulted from Case b or Case c always has a good triangulation and one can thus be constructed in constant time since R has at most six edges. In summary, the improvement still runs with $O(n^2)$ storage and $O(n^2 \log n)$ time.

Reducing Angle Bound. The angle bound of $\frac{3}{4}\pi$ can slightly be improved to $\alpha = \frac{11}{15}\pi$ (i.e. 132^o), but with a larger constant in the quadratic bound on vertex size. In this case, isosceles triangles has base angles $\beta = \frac{4}{15}\pi$ (i.e. 48^o) and apex angle $\phi = \frac{7}{15}\pi$ (i.e. 84^o). All results developed starting from Section 4 are valid except for Lemma 4.1 and Lemma 9.2. We need not worry about Lemma 9.2 since it is no longer relevant to the above improved construction. As for Lemma 4.1, problem arises when $\phi < \angle p < 2\beta$. One way to resolve this is to perform the following before Step 1: subdivide each triangle pqr of $\mathcal{T}$ with largest angle less than 2β by a new vertex s inside pqr into triangles with angles at s equal to $\frac{2}{3}\pi$. The existence of such a subdivision can easily be verified.

11 Concluding Remarks

Given a plane geometric graph, this paper shows that a conforming triangulation with a quadratic bound on its vertex set and the $\frac{3}{4}\pi$ bound on its angles can be computed in slightly more than quadratic time. Additionally, the result applies to the better angle bound of $\frac{11}{15}\pi$, but with a larger constant factor in the quadratic bound on the vertex set. This paper improves the result of Mitchell [Mitc93b] by reducing the asymptotic bound on the vertex set by a logarithmic factor and the angle measure by at least $\frac{\pi}{8}$. Further reducing the angle bound closer to $\frac{\pi}{2}$ is still very interesting.

The main idea of the paper is on the control of the lengths of propagation paths using fences and traps. This is a similar strategy used in Edelsbrunner and Tan [EdTa93], where the corresponding concept is termed *walls*. The current paper is, however, much more complex; major difficulties include the construction of the linear number of traps and the modification of propagation paths, whereas these issues do not appear in [EdTa93] due to the nice feature of constructing walls inside disks that are mutually disjoint and using only non-crossing propagation paths.

Acknowledgement

I would like to thank Scott Mitchell for providing details of his paper [Mitc93b].

References

[BaAz76] I. Babuška and A. K. Aziz. On the angle condition in the finite element method. *SIAM J. Numer. Anal.* **13** (1976), 214–226.

[BaLi84] R. E. Barnhill and F. F. Little. Three and four-dimensional surfaces. *Rocky Mountain J. Math.* **14** (1984), 77–102.

[BDE92] M. Bern, D. Dobkin and D. Eppstein. Triangulating polygons without large angles. *In* "Proc. 8th Ann. Sympos. Comput. Geom., 1992", 222–231.

[BEG90] M. Bern, D. Eppstein and J. Gilbert. Provably good mesh generation. *In* "Proc. 31st Ann. IEEE Sympos. Found. Comput. Sci., 1990", 231–241. To appear in *J. Comp. System Science.*

[BeEp91] M. Bern and D. Eppstein. Polynomial-size nonobtuse triangulation of polygons. *Intl. J. Comput. Geom. Appl.* **2(3)** (1992), 241–255.

[BeEp92] M. Bern and D. Eppstein. Mesh Generation and Optimal Triangulation. In *Computing in Euclidean Geometry*, D. Z. Du and F. K. Hwang, eds., World Scientific, Singapore, 1992, 23–90.

[BMR94] M. Bern, S. A. Mitchell and J. Ruppert. Linear-size non-obtuse triangulation of polygons. *In* "Proc. 10th Ann. Sympos. Comput. Geom., 1994".

[Bois88] J. D. Boissonnat. Shape reconstruction from planar cross sections. *Computer Vision, Graphics, and Image Process.* **44** (1988), 1–29.

[Cave74] J. Cavendish. Automatic triangulation of arbitrary planar domains for the finite element method. *Intl. J. Numer. Methods Engrg.* **8** (1974), 679–696.

[Dey91] T. K. Dey. Decompositions of Polyhedra in Three Dimensions. Techn. Rep. CSD-TR-91-056, Ph.D. Thesis, Comput. Sci. Dept., Purdue Univ., IN, 1991.

[EdTa93] H. Edelsbrunner and T. S. Tan. An upper bound for conforming Delaunay triangulations. *Discrete Comput. Geom.* **10** (1993), 197–213.

[ETW92] H. Edelsbrunner, T. S. Tan, and R. Waupotitsch. An $O(n^2 \log n)$ time algorithm for the minmax angle triangulation. *SIAM J. Sci. Statist. Comput.* **13** (1992), 994–1008.

[Greg75] J. A. Gregory. Error bounds for linear interpolation on triangles. *The Mathematics of Finite Element and Applications II*, J. R. Whiteman, ed., Academic Press, NY, 1975, 163–170.

[Laws77] C. L. Lawson. Software for C^1 surface interpolation. In *Math. Software III*, J. R. Rice, ed., Academic Press, 1977, 161–194.

[MeSo92] E. A. Melissaratos and D. L. Souvaine. Coping with inconsistencies: a new approach to produce quality triangulations of polygonal domains with holes. *In* "Proc. 8th Ann. Sympos. Comput. Geom., 1992", 202–211.

[Mitc93a] S. A. Mitchell. Mesh Generation with Provable Quality Bounds. Techn. Rep. CS-TR-92-1327, Ph.D. Thesis, Comput. Sci. Dept., Cornell Univ., NY, 1993.

[Mitc93b] S. A. Mitchell. Refining a triangulation of a planar straight-line graph to eliminate large angles. *In* "Proc. 34th Ann. IEEE Sympos. Found. Comput. Sci., 1993", 583–591.

[Mitc94] S. A. Mitchell. Finding a covering triangulation whose maximum angle is provably small. *In* "Proc. 17th Ann. Computer Science Conference, ACSC-17, 1994".

[Rupp93] J. Ruppert. A new and simple algorithm for quality 2-dimensional mesh generation. *In* "Proc. 4th ACM-SIAM Sympos. Discrete Algorithms, 1993", 83–92.

[Tan93] T. S. Tan. Optimal Two-Dimensional Triangulations. Techn. Rep. UIUCDCS-R-92-1783, Ph.D. Thesis, Comput. Sci. Dept., Univ. Illinois, Urbana, IL, 1993.

[Tan94] T. S. Tan. An optimal bound for conforming quality triangulations. Techn. Rep., Dept. Inform. Sys. & Comput. Sci., National Univ. Singapore, Singapore, 1994.

On the Maximum Degree of Minimum Spanning Trees

Gabriel Robins and Jeffrey S. Salowe*

Department of Computer Science, University of Virginia, Charlottesville, VA 22903-2442

Abstract

Motivated by practical VLSI routing applications, we study the maximum vertex degree of a minimum spanning tree (MST). We prove that under the L_p norm, the maximum vertex degree over all MSTs is equal to the Hadwiger number of the corresponding unit ball; we show an even tighter bound for MSTs where the maximum degree is minimized. We give the best-known bounds for the maximum MST degree for arbitrary L_p metrics in all dimensions, with a focus on the rectilinear metric in two and three dimensions. We show that for any finite set of points in the Manhattan plane there exists an MST with maximum degree of at most 4*, and for three-dimensional Manhattan space the maximum possible degree of a minimum-degree MST is either* 13 *or* 14.

1 Introduction

Minimum spanning tree (MST) construction is a classic optimization problem for which several efficient algorithms are known [8] [14] [17]. Solutions of many other problems hinge on the construction of an MST as an intermediary step [3], with the time complexity sometimes depending exponentially on the MST's maximum vertex degree, as in the algorithm of Georgakopoulos and Papadimitriou [7]. Applications that would benefit from MSTs with low maximum vertex degree include Steiner tree approximation [13] as well as VLSI global routing [1] [16]. With this in mind, we seek efficient methods to construct an MST with low maximum vertex degree:

The Bounded Degree Minimum Spanning Tree (BDMST) problem: Given a complete weighted graph and an integer $D \geq 2$, find a minimum-cost spanning tree with maximum vertex degree $\leq D$.

Finding a BDMST of maximum degree $D = 2$ is equivalent to solving the traveling salesman problem, which is known to be NP-hard [6]. Papadimitriou and Vazirani have shown that that given a planar pointset, the problem of finding a Euclidean BDMST with $D = 3$ is also NP-hard [15]. On the other hand, they also note that a BDMST with $D = 5$ in the Euclidean plane is actually a Euclidean MST and can therefore be found in polynomial time. The complexity of the BDMST problem when $D = 4$ remains open. Ho, Vijayan, and Wong [11], proved that an MST in the rectilinear plane must have maximum degree of $D \leq 8$, and state (without proof) that the maximum degree bound may be improved to $D \leq 6$. The results of Guibas and Stolfi [9] also imply the $D \leq 8$ bound.

In this paper we settle the bounded-degree MST problem in the rectilinear plane: we show that given a planar pointset, the rectilinear BDMST problem with $D \leq 3$ is NP-hard, but that the rectilinear $D \geq 4$ case is solvable in polynomial time. In particular, we prove that in the rectilinear plane there always exists an MST with maximum degree of $D \leq 4$, which is tight. We also analyze the maximum MST degree in three dimensions under the rectilinear metric, which arises in three-dimensional VLSI applications [10], where we show a lower bound of 13 and an upper bound of 14 on the maximum MST degree (using the previously known techniques, the best obtainable lower and upper bounds for three dimensions under the rectilinear metric were 6 and 26, respectively).

More generally, for arbitrary dimension and L_p metrics we investigate : (i) the maximum possible vertex degree of an MST, and (ii) the maximum degree of MSTs in which the maximum degree is minimized.

*This work was partially supported by NSF grants MIP-9107717 and CCR-9224789. The authors' Email addresses are robins@cs.virginia.edu and salowe@cs.virginia.edu.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

We relate the maximum MST degree under the L_p metric to the so-called Hadwiger number of the corresponding L_p unit ball. The relation between MST degree and the packing of convex sets has not been elucidated before, though Day and Edelsbrunner [5] studied the related "attractive power" of a point. For general dimension we give exponential lower bounds on the Hadwiger number and on the maximum MST degree.

Our results have several practical applications. For example, our efficient algorithm to compute an MST with low maximum degree enables an efficient implementation of the Iterated 1-Steiner algorithm of Kahng and Robins for VLSI routing [13], which affords a particularly effective approximation to a rectilinear Steiner minimal tree (within 0.5% of optimal for typical input pointsets [18]), and where the central time-consuming loop depends on the maximum MST degree [1] [2]. Our results also have implications to newly emerging three-dimensional VLSI technologies [10], as well as for any other algorithms that use an MST as the basis for some other construction.

The remainder of the paper is as follows. Section 2 establishes the terminology and relates the maximum MST degree to the Hadwiger numbers (the central result is Theorem 4). Section 3 studies the L_1 Hadwiger numbers and the maximum MST degree for the L_1 metric. Section 4 considers the Hadwiger numbers and the maximum MST degree for arbitrary L_p metrics. We conclude in Section 5 with open problems.

2 Hadwiger and MST Numbers

A collection of open convex sets forms a *packing* if no two sets intersect; two sets that share a boundary point in the packing are said to be *neighbors*. The *Hadwiger number* $H(B)$ of an open convex set B is the maximum number of neighbors of B considered over all packings of translates of B (a *translate* of B is a congruent copy of B moved to another location in space while keeping B's original orientation).

There is a vast literature on Hadwiger numbers (e.g., Croft et al. [4], Fejes Tóth [19]). Most results address the plane, but there are several results for higher dimensions. In particular, if S is a convex set in $\Re^k$ (i.e., k-dimensional space), the Hadwiger number $H(S)$ satisfies:

$$k^2 + k \leq H(S) \leq 3^k - 1$$

It is known that the regular k-simplex realizes the lower bound and the k-hypercube realizes the upper bound [19]. Tighter bounds are known for k-hyperspheres; Wyner [20] showed that the Hadwiger number for spheres is at least $2^{0.207k(1+o(1))}$, and Kabatjansky and Levenštein [12] showed that it is at most $2^{0.401k(1+o(1))}$. Only four Hadwiger numbers for spheres are known exactly; these are the numbers in dimensions 2, 3, 8, and 24, and they are 6, 12, 240, and 196560, respectively. The three-dimensional Hadwiger number has a history dating back to Newton and was only determined much later.

For two points $x = (x_1, x_2, \ldots, x_k)$ and $y = (y_1, y_2, \ldots, y_k)$ in k-dimensional space $\Re^k$, the L_p distance between x and y is $||xy||_p = \sqrt[p]{\sum_{i=1}^{k} |x_i - y_i|^p}$. For convenience, if the subscript p is omitted, the rectilinear metric is assumed (i.e., $p = 1$). Let $B(k,p,x)$ denote an open L_p unit ball centered at a point x in $\Re_p^k$ (we use $\Re_p^k$ to denote k-dimensional space where distances are computed under the L_p metric). When x is the origin, $B(k,p,x)$ is denoted as $B(k,p)$, and we use $H(k,p)$ to denote the Hadwiger number of $B(k,p)$.

Let $I(k,p)$ be the maximum number of points that can be placed on the boundary of an L_p k-dimensional unit ball so that each pair of points is at least a unit apart. With respect to a finite set of points $P \subset \Re_p^k$ and an MST T for P, let $\nu(k,p,P,T)$ be the maximum vertex degree of T (also referred to as simply the degree of T). Let $\nu(k,p,P) = \max\limits_{T \in \tau} \nu(k,p,P,T)$, where τ is the set of all MSTs for P, and let $\nu(k,p) = \max\limits_{P \subset \Re^k} \nu(k,p,P)$. In other words, $\nu(k,p,P)$ denotes the maximum degree of any MST over P, and $\nu(k,p)$ denotes the highest possible degree in any MST over any finite pointset in k-dimensional space under the L_p metric.

We will need the following result, which is easily established.

Lemma 1 $||xy||_p \geq 2$ *if and only if* $B(k,p,x) \cap B(k,p,y) = \emptyset$. *Further,* $||xy||_p = 2$ *if and only if* $B(k,p,x)$ *and* $B(k,p,y)$ *are tangent.*

Lemma 2 *The Hadwiger number* $H(k,p)$ *is equal to* $I(k,p)$*, the maximum number of points that can be placed on the boundary of the unit ball* $B(k,p)$ *so that all interpoint distances are at least one unit long.*

Proof: We first show that $I(k,p) \leq H(k,p)$. Suppose that $I(k,p)$ points are on the boundary of $B(k,p)$ and are each at least a unit distance apart. Consider placing an L_p ball of radius $\frac{1}{2}$ around each point, including one at the origin. By Lemma 1, these balls form a packing, and all the balls touch the ball containing the origin. Therefore, $I(k,p) \leq H(k,p)$.

Next we show that $H(k,p) \leq I(k,p)$. Consider a packing of L_p unit balls, and choose one to be centered at the origin. Consider the edges connecting the origin to each center of the neighboring balls. The intersections of these edges with the boundary of $B(k,p)$ yield a pointset where each pair of points is separated by at least a unit distance, otherwise we would not have a packing of L_p unit balls. □

Lemma 3 *The Hadwiger number $H(k,p)$ is equal to $\nu(k,p)$, the maximum MST degree over any finite pointset in $\Re_p^k$.*

Proof: We show that $\nu(k,p) \leq I(k,p)$, the maximum number of points that can be placed on the boundary of an L_p k-dimensional unit ball so that each pair of points is at least a unit apart. Let x be a point, and let $y_1, \ldots, y_\nu$ be points adjacent to x in an MST, indexed in order of increasing distance from x. Note that (y_i, y_j) must be a longest edge in the triangle (x, y_i, y_j), and that the MST restricted to x and $y_1, \ldots, y_\nu$ is a star centered at x. Draw a small L_p ball around x, without loss of generality a unit ball, and consider the intersection of the segments (x, y_i), $1 \leq i \leq \nu$, with $B(k,p,x)$. Let these intersection points be called $\hat{y}_i$, $1 \leq i \leq \nu$, and suppose there is a pair $\hat{y}_i$ and $\hat{y}_j$, $i < j$, with $||\hat{y}_i \hat{y}_j||_p \leq 1$. Note that $(\hat{y}_i, \hat{y}_j)$ is the shortest edge on the triangle $(x, \hat{y}_i, \hat{y}_j)$, and $(x, \hat{y}_j)$ is a longest edge. Now consider similar triangle (x, y_i, z), where z is a point on the edge (x, y_j). The path from y_j to z to y_i is shorter than the length of (x, y_j), so (y_i, y_j) is not a longest edge in triangle (x, y_i, y_j), a contradiction. We note this bound is tight for pointsets that realize $I(k,p)$. □

We next consider a slightly different number $\hat{\nu}(k,p)$, which is closely related to $\nu(k,p)$. Recall that $\nu(k,p,P,T)$ denotes the maximum vertex degree of the tree T. Let $\hat{\nu}(k,p,P) = \min_{T \in \tau} \nu(k,p,P,T)$, where τ is the set of all MSTs for P, and let $\hat{\nu}(k,p) = \max_{P \subset \Re^k} \nu(k,p,P)$. In other words, $\hat{\nu}(k,p,P)$ denotes the degree of an MST over P that has the smallest possible degree, and $\hat{\nu}(k,p)$ denotes the maximum of the degrees of all minimum-degree MSTs over all finite pointsets in $\Re_p^k$ (recall that we use the phrase "the degree of T" to refer to the maximum vertex degree of the tree T).

Although it is clear that $\hat{\nu}(k,p) \leq \nu(k,p)$, it is not clear when this inequality is strict. In order to count $\hat{\nu}(k,p)$, we define the MST number $M(k,p)$ similarly to the Hadwiger number $H(k,p)$, except that the translates of the L_p unit ball $B(k,p)$ are slightly magnified. The underlying packing consists of $B(k,p)$ as well as multiple translated copies of $(1+\epsilon) \cdot B(k,p)$, and $M(k,p)$ is the supremum over all $\epsilon > 0$ of the maximum number of neighbors of $B(k,p)$ over all such packings. Clearly $M(k,p) \leq H(k,p)$. We also define $\hat{I}(k,p)$ as the number of points that can be placed on the boundary of $B(k,p)$ so that each pair is strictly greater than one unit apart.

Consider a set $S = \{x_1, \ldots, x_n\}$ of n points in $\Re_p^k$. For convenience, let $N = 2^{\binom{n}{2}}$, and let $S_1, \ldots, S_N$ be the set of sums of the interdistances, one sum for each distinct subset. Let

$$0 < \delta = \min_{1 \leq i < j \leq N} \{|S_i - S_j| : |S_i - S_j| > 0\}$$

A *perturbation* of a pointset S is a bijection from S to a second set $S' = \{x'_1, \ldots, x'_n\}$ (for convenience, assume that the indices indicate the bijection); we say that a perturbation of S is *small* if

$$\sum_{i=1}^{n} ||x_i x'_i||_p < \frac{\delta}{2}$$

In discussing spanning trees of S and the perturbed set S', we assume that the vertex set $[n]$ consists of the integers 1 to n, where vertex i corresponds to point x_i or point x'_i. The *topology* of a tree over vertex set $[n]$ is the set of edges in the tree.

Theorem 4 *Let S be a set of points in $\Re_p^k$, and let S' be a set of points corresponding to a small perturbation of S. Then the topology of an MST for S' is also a topology for an MST for S.*

Proof: Let T be an MST for S, and let T' be an MST for S'. Let $l(T)$ and $l(T')$ be the lengths of T and T', respectively. Then $l(T) - \frac{\delta}{2} < l(T') < l(T) + \frac{\delta}{2}$. Consider the tree $\hat{T}$ with the same topology as T' but with respect to pointset S. Now, $l(T') - \frac{\delta}{2} < l(\hat{T}) < l(T') + \frac{\delta}{2}$, so $l(T) - \delta < l(\hat{T}) < l(T) + \delta$. Since δ is the minimum positive difference between the sums of any two distinct subsets of interdistances, $l(T) = l(\hat{T})$, and $\hat{T}$ is also an MST for S. □

Lemma 5 *The maximum of the degrees of all minimum-degree MSTs over all finite pointsets in $\Re_p^k$, is equal to the maximum number of slightly magnified unit balls that can be packed around a given unit ball; that is, $\hat{\nu}(k,p) = M(k,p)$.*

Proof: Let S be a set of points, and let δ be defined as above. Place a small L_p ball about each point $x \in S$ (without loss of generality a unit ball, though the intent is that x is the only point inside $B(k,p,x)$), and connect each distinct pair (x,y), $x,y \in S$, with a line segment. Consider the intersections of these edges with $B(k,p,x)$. Perform a small perturbation on S so that no two intersection points have length 1. Repeat the argument used in the proof of Lemmas 2 and 3, this time with balls of the form $(1+\epsilon) \cdot B(k,p)$, for small $\epsilon > 0$. The first part shows that $\hat{I}(k,p) = M(k,p)$, and the second that $\hat{\nu}(k,p) \leq \hat{I}(k,p)$. This bound is tight for pointsets that realize $\hat{I}(k,p)$. □

3 The Maximum L_1 MST Degree

Hadwiger numbers are notoriously difficult to compute. In this section, we determine the 2 and 3 dimensional Hadwiger numbers for the diamond and octahedron, respectively. The first of these numbers is well-known, but we could not find any reference for the octahedron. We also study the MST numbers, obtaining a value of 4 in two dimensions, and bounds in higher dimensions. For notational convenience, we define:

The Uniqueness Property: Given a point p, a region R has the *uniqueness property* with respect to p if for every pair of points $u, w \in R$, $||wu| < \max(||wp||, up||)$.

A partition of space into a finite set of disjoint regions is said to have the uniqueness property if each of its regions has the uniqueness property.

Define the *diagonal partition* of the plane as the partition induced by the two lines oriented at 45 and -45 degrees through a point p (i.e., partitioning $\Re^2 - \{p\}$ into 8 disjoint regions, four "wedges" of dimension two (labeled $R1$ through $R4$ in Figure 1(a)), and four "half-lines" of dimension one (labeled $R5$ through $R8$ in Figure 1(a)). It is easy to show that the diagonal partition has the uniqueness property, which in turn implies an upper bound of 8 on the maximum MST degree in the Manhattan plane.

Lemma 6 *Given a point p in the Manhattan plane, each region of the diagonal partition with respect to p has the uniqueness property.*

Proof: We need to show that for any two distinct points u and w that lie in the same region, $||wu| < \max(||wp||, up||)$. This is obvious for the one dimensional regions. Consider u, w in one of the two dimensional regions. Assume without loss of generality that $||up|| \leq ||wp||$ (otherwise swap the roles of u and v in this proof). Consider the diamond D with left corner at p and center at c, such that u is on the boundary of D (see Figure 1(c)). Let a ray starting at p and passing through w intersect D at b. By the triangle inequality, $||wu|| \leq ||wb|| + ||bu|| < ||wb|| + ||bc|| + ||cu|| = ||wb|| + ||bc|| + ||cp|| = ||wp||$. Thus every one of the 8 regions of the diagonal partition has the uniqueness property. □

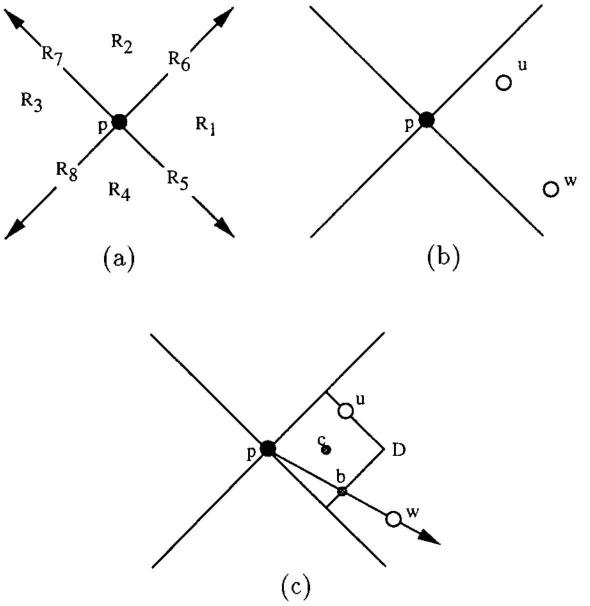


Figure 1: A partition of $\Re - \{p\}$ into 8 regions such that for any two points u and w that lie in the same region, either $||wu|| < ||wp||$ or else $||uw|| < ||up||$.

Corollary 7 *The maximum possible degree of an MST over any finite pointset in the rectilinear plane is equal to* 8; *that is,* $\nu(2,1) = 8$.

Proof: By Lemma 6 the diagonal partition has the uniqueness property, which implies that $I(2,1) \leq 8$. The pointset $\{(0,0), (\pm 1, 0), (0, \pm 1), (\pm\frac{1}{2}, \pm\frac{1}{2})\}$ shows that this bound is tight. Lemmas 2 and 3 imply that the maximum MST degree in the Manhattan plane is equal to 8. □

We now show an analogous result for three-dimensional Manhattan space. Consider a *cuboctahedral* partition of $\Re^3_1$ into 14 disjoint regions corresponding to the faces of a truncated cube (Figure 2(a-b)), i.e., 6 congruent pyramids with *square* cross-section (Figure 2(c)) and 8 congruent pyramids with *triangular* cross-section (Figure 2(d)). Most of the region boundaries are included into the triangular pyramid regions as shown in Figure 3(c), with the remaining boundaries forming 4 new regions (Figure 3(d)), to a total of 18 regions. Following the same strategy as in the two-dimensional case, we first show that the uniqueness property holds.

Lemma 8 *Given a point p in three-dimensional Manhattan space, each region of the cuboctahedral partition with respect to p has the uniqueness property.*

Proof: We need to show that for any two points u and w that lie in the same region of the cuboctahedral partition, $||wu| < \max(||wp||, up||)$. This is obvious for the 2 dimensional regions that are the boundaries between the pyramids (by an argument analogous to that of Lemma 6).

Consider one of the square pyramids R with respect to p (Figure 2(c)), and let $u, w \in R$. Assume without loss of generality that $||up|| \leq ||wp||$ (otherwise swap the roles of u and w). Consider the locus of points $D \subset R$ that are distance $||up||$ from p. (Figure 2(e)); D is the upper half of the boundary of an octahedron. Let c be the center of the octahedron determined by D, so that c is equidistant from all points of D. Let b be the intersection of the surface of D with a ray starting from p and passing through w. By the triangle inequality, $||wu|| \leq ||wb|| + ||bu|| < ||wb|| + ||bc|| + ||cu|| = ||wb|| + ||bc|| + ||cp|| = ||wp||$ (recall that the square pyramid regions do not contain their boundary points). Thus, w is closer to u than it is to p, which implies that the region R has the uniqueness property.

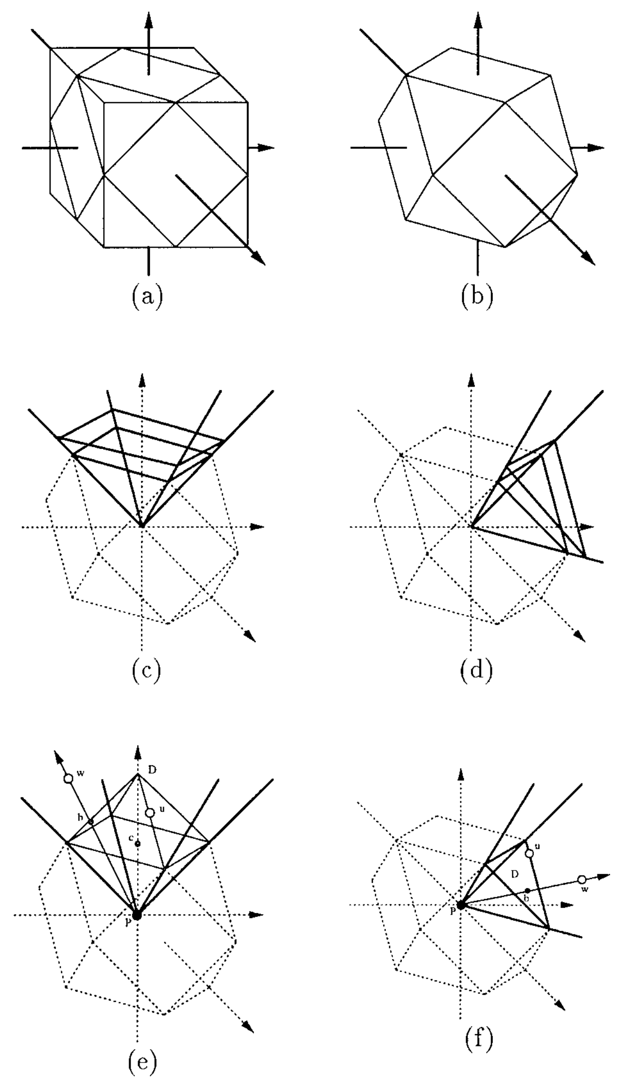


Figure 2: A truncated cube (a-b) induces a 3-dimensional cuboctahedral partition of space into 14 regions: 6 square pyramids (c), and 8 triangular pyramids (d). Using the triangle inequality, each region may be shown to contain at most one candidate point for connection with the origin in an MST (e-f).

To show the uniqueness property for the triangular pyramids, consider one of the triangular pyramids R with respect to p (Figure 2(d)), and let $u, w \in R$. Assume without loss of generality that $||up|| \leq ||wp||$ (otherwise swap the roles of u and w). Consider the locus of points D in R that are at distance $||up||$ from p (Figure 2(f)). Let b be the intersection of D with a ray

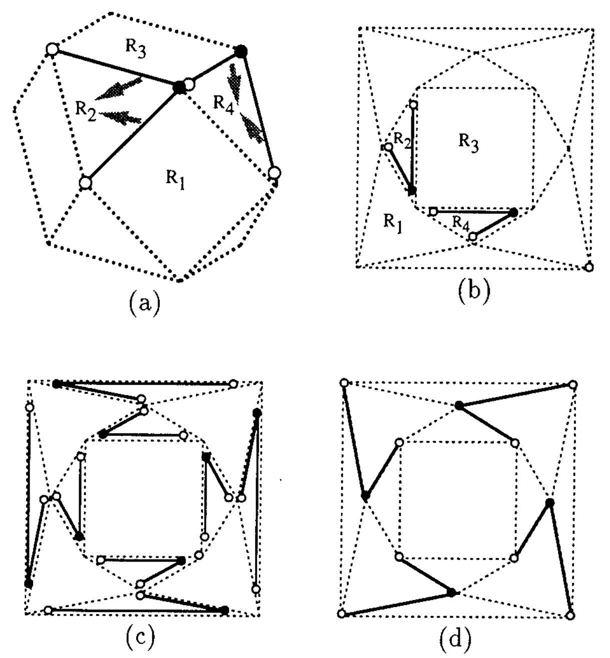


Figure 3: Assigning the boundary points to the various region: a hollow (solid) dot indicates an open (closed) interval. A topological mapping of the cuboctahedron (a) is shown in (b). The various boundaries are included into the triangular pyramid regions (c), while the square pyramids do not contain any boundary points. The remaining boundaries form 4 new regions (d), bringing the total to 18.

starting from p and passing through w. By the triangle inequality, $||wu|| \leq ||wb||+||bu|| < ||wb||+||bp|| = ||wp||$ (recall that each triangular region is missing one of its boundary faces, as shown in Figure 3(a-b)). Thus, w is closer to u than it is to p, which implies that the region R has the uniquenes property.

Thus every one of the 18 regions of the cuboctahedral partition has the uniqueness property. □

Corollary 9 *The maximum possible degree of an MST over any finite pointset in three-dimensional Manhattan space is equal to* 18*; that is,* $\nu(3,1) = 18$.

Proof: By Lemma 8 the cuboctahedral partition has the uniqueness property, which implies that $I(3,1) \leq 18$. The pointset $\{(0,0,0), (\pm 1,0,0), (0,\pm 1,0), (0,0,\pm 1), (\pm\frac{1}{2},\pm\frac{1}{2},0), (0,\pm\frac{1}{2},\pm\frac{1}{2}), (\pm\frac{1}{2},0,\pm\frac{1}{2})\}$ shows that this bound is tight. Lemmas 2 and 3 imply that the maximum MST degree in three-dimensional Manhattan space is 18. □

We can further refine the maximum MST degree bound of Corollary 7 by applying the perturbative argument of Theorem 4.

Theorem 10 *The maximum degree of a minimum-degree MST over any finite pointset in the rectilinear plane is* 4*; that is,* $\hat{\nu}(2,1) = 4$.

Proof: The pointset $\{(0,0), (\pm 1,0), (0,\pm 1)\}$ establishes a lower bound of 4. To get the upper bound of 4, consider $\hat{I}(2,1)$, the number of points that can be placed on boundary of a unit ball (i.e., a diamond) in $\Re_1^2$ such that each pair of points is strictly greater than one unit apart. Consider the diagonal partition, as in the proof of Lemma 6; at most one point can be in the closure of each of the four 2-dimensional regions, proving the result. □

Theorem 10 has an interesting consequence on the complexity of the BDMST problem restricted to the rectilinear plane:

Instance: A planar pointset $P = \{x_1, \ldots, x_n\}$, and integers D and C.

Question: Is there a rectilinear spanning tree with maximum degree $\leq D$ and cost $\leq C$?

If $D = 4$, the question can be decided in polynomial time. In fact, our methods establish that such a bounded-diameter MST can be computed as efficiently as an ordinary MST. On the other hand, if $D = 2$, the problem is essentially a rectilinear traveling salesman problem (a *wandering* salesman problem, since the tour is a path rather than a circuit), and it is therefore NP-complete. It turns out that the $D = 3$ question is also NP-complete, using a proof identical to the one appearing in Papadimitriou and Vazirani [15] (their result is for the corresponding Euclidean problem, but since they restrict their construction to a special type of *grid graph*, their proof holds in the rectilinear metric as well). We can summarize the rectilinear and Euclidean results in the following table:

Complexity of the BDMST Problem		
D	Euclidean	rectilinear
2	NP-complete	NP-complete
3	NP-complete	NP-complete
4	open	polynomial
≥ 5	polynomial	polynomial

Next, we refine the maximum MST degree bound of Corollary 9.

Theorem 11 *The maximum degree of a minimum-degree MST over any finite pointset in* 3*-dimensional rectilinear space is either* 13 *or* 14*; that is,* $13 \leq \hat{\nu}(3,1) \leq 14$.

Proof: For the lower bound, the following pointset shows that the maximum degree of an MT is $\hat{I}(k,p) \geq 13$: $\{(0,0,0)$, $(\pm 100,0,0)$, $(0,\pm 100,0)$, $(0,0,\pm 100)$, $(47,-4,49)$, $(-6,-49,45)$, $(-49,8,43)$, $(-4,47,-49)$, $(-49,-6,-45)$, $(8,-49,-43)$, $(49,49,2)\}$, since for this pointset, all non-origin points are strictly closer to the origin than they are to each other, forcing the MST to be unique with a star topology.

To obtain the upper bound of 14 on the maximum MST degree, consider the cubeoctahedral partition. Any two points lying in the closure of one of the 14 main regions of the cuboctahedral partition must be within distance 1 of each other. □

We note that there is an elementary means to settle the 13 vs. 14 question raised in Theorem 11. Suppose we are trying to decide whether 14 points can be placed on the surface of a unit octahedron so that each pair is greater than a unit distance apart. The relationship between point (x_i, y_i, z_i) and point (x_j, y_j, z_j) can be phrased by the inequality $|x_i - x_j| + |y_i - y_j| + |z_i - z_j| > 1$ subject to the constraints $|x_i| + |y_i| + |z_i| = 1$ and $|x_j| + |y_j| + |z_j| = 1$. The absolute values can be removed if the relative order between x_1 and x_2, etc., is known. We can therefore consider all permutations of the coordinates of the 14 points and produce the corresponding inequalities. If the inequalities corresponding to a particular permutation are simultaneously satisfied, $\hat{\nu}(3,1) = 14$, otherwise $\hat{\nu}(3,1) = 13$. Feasibility can be settled by determining whether a particular polytope contains a nonempty relative interior (this approach is easily extended to any dimension k). We have not settled the 13 vs. 14 question, and the above procedure seems impractical due to the large number of resulting inequalities.

We now address the Hadwiger and MST Numbers for the k-crosspolytope.

Theorem 12 *The maximum degree of a minimum-degree MST over any finite pointset in* $\Re_1^k$ *is at least* $\Omega(2^{0.0312k})$*; that is,* $\hat{\nu}(k,1) = \Omega(2^{0.0312k})$.

Proof: Consider the family $F(j)$ of points $(\pm\frac{1}{j}, \ldots, \pm\frac{1}{j}, 0, \ldots, 0)$, where j is an integer between 1 and k. (Here, the j nonzero terms can be arbitrarily interspersed in the vector.)

Each member of $F(j)$ is distance 1 from the origin; the distance between $x \in F(j)$ and $y \in F(j)$ depends on the positions and signs of the nonzero terms. Given $x = (x_1, x_2, \ldots, x_k) \in F(j)$, let $\bar{x}$ be the binary vector containing a 1 in bit i if $x_i \neq 0$ and a 0 in bit i if $x_i = 0$. If the Hamming distance between $\bar{x}$ and $\bar{y}$ is at least j, then $||xy|| \geq 1$. (The Hamming distance between two bit vectors is the number of bit positions in which they differ.) We want to find a large set of $\bar{x}$ that are mutually Hamming distance greater than j apart.

Consider the set $V(j)$ of bit vectors containing exactly j 1's; $|V(j)| = \binom{k}{j}$. Form a graph $G(t) = (V(t), E(t))$ for which $(\bar{x}, \bar{y}) \in E(t)$ if and only if the Hamming distance between $\bar{x}$ and $\bar{y}$ is at most j. Note that $G(t)$ is regular with degree $d(j) = \sum_{i=1}^{\lfloor j/2 \rfloor} \binom{j}{i}\binom{k-j}{i}$. To see this, we determine the number of edges adjacent to $\bar{x} = (1, \ldots, 1, 0, \ldots, 0)$, where there are j 1's. The set of vectors in $V(j)$ adjacent to $\bar{x}$ can be partitioned into vectors that contain i 0's in the first j positions, $1 \leq i \leq \lfloor \frac{j}{2} \rfloor + 1$. For a given i, there are $\binom{j}{i}$ ways to choose the 0 positions and $\binom{k-j}{i}$ positions to place the displaced 1's in the last $k - j$ positions.

Here is our strategy to find a subset of $V(j)$ of large cardinality that are mutually far apart: choose a vertex, delete its neighbors, and continue. The number of vertices chosen must exceed $|V(j)|/d(j)$. Suppose that $cj = 16\sqrt{e}j = k$. Then

$$\frac{|V(j)|}{d(j)} \geq \frac{\binom{k}{j}}{(\frac{j}{2}+1)\binom{j}{\frac{j}{2}}\binom{k-j}{\frac{j}{2}}} \geq \frac{\binom{k}{j}}{(\frac{j}{2}+1)\frac{4^j}{\sqrt{\pi j}}\binom{k-j}{\frac{j}{2}}}$$

$$> c' \frac{c^{j/2}}{4^j \sqrt{j}} = c'' \left(\frac{c}{16}\right)^{\frac{k}{2c}} \frac{1}{\sqrt{k}}$$

Here, c' and c'' are constants; the approximation to $\binom{j}{\frac{j}{2}}$ is from Graham et al. [8]. Substituting for c gives the result. □

4 The Maximum L_p MST Degree

In this section, we provide bounds on $M(k,p)$ for general L_p metrics.

Theorem 13 *The maximum degree of a minimum-degree MST over any finite pointset in $\Re_p^k$ is at least $\hat{\nu}(k,p) = \Omega(\sqrt{k}2^{n(1-E(\alpha))})$, where $\alpha = \frac{1}{2^p}$ and $E(x) = x \lg \frac{1}{x} + (1-x) \lg \frac{1}{1-x}$.*

Proof: Consider the vertices of the k-hypercube $(\pm 1, \ldots, \pm 1)$. Each of these points is $k^{1/p}$ from the origin. On the other hand, if points x and y differ from each other in j positions, they are distance $2j^{1/p}$ from each other. If $2j^{1/p} > k^{1/p}$, then x and y are further from each other than they are from the origin.

We need to find the largest cardinality set of points on the k-hypercube that differ in at least $J = \frac{k}{2^p}$ positions. To do this, construct a graph G whose vertex set is the set of binary strings of length k, and for which there is an edge between string a and string b if and only if the Hamming distance between a and b is at most J. Proceed in the same manner as in the proof of Theorem 12, except that $d(J) = \sum_{i=1}^{J} \binom{k}{i}$. The number of vertices chosen must exceed $2^k/d(J)$. Now,

$$\sum_{i \le \alpha k} \binom{k}{i} = 2^{kE(\alpha) - \frac{1}{2}\lg k + O(1)}$$

for $0 < \alpha < \frac{1}{2}$ (see Graham *et al.* [8], Chapter 9, Problem 42). Note that $\alpha = \frac{1}{2^p}$, so

$$\frac{2^k}{d(J)} = \sqrt{k}2^{k(1-E(2^{-p}))}$$

□

Theorem 13 shows that for any fixed $p > 1$, $\hat{\nu}(k,p)$ grows exponentially in the dimension. Note that this bound is less than the bound obtained by Wyner [20] (for $H(k,2)$, it is $\Omega(2^{0.189k})$ since $E(\frac{1}{4}) = \frac{3}{4}\lg 3 - 1 \approx 0.189$), but it is sufficient for our purposes.

It is well known that $H(k,p) \le 3^k - 1$ (e.g., [19]). In 2-dimensional space, the Hadwiger number is largest for L_1 and L_∞, the only planar L_p metrics with Hadwiger number 8. For all other L_p metrics, the Hadwiger number is 6. On the other hand, the planar MST number is smallest for L_1 and L_∞, having a value of 4, and it is easily seen to be 5 for all other L_p metrics.

These observations raise an interesting question: how does the MST number behave as a function of p? Note that the maximum Hadwiger number is achieved by parallelotopes. Next we derive the MST number for the L_∞ unit ball (i.e., the k-hypercube), and show that the MST number is not maximized in the L_∞ metric in any dimension.

Theorem 14 *The maximum degree of a minimum-degree MST over any finite pointset in $\Re_\infty^k$ is 2^k; that is, $\hat{\nu}(k,\infty) = 2^k$.*

Proof: We first show the result for $p = \infty$; note that the L_p unit ball is a k-hypercube. The upper bound is established by considering $\hat{I}(k,p)$, the number of points that can be placed on boundary of a unit ball in $\Re_p^k$ such that each pair of points is strictly greater than one unit apart. Note that at most one point can be placed in each k-ant (the k-dimensional analogue of "quadrant"). The lower bound is established by considering the set of 2^k vertices of a k-hypercube. □

Theorem 15 *For each k, there is a p such that the maximum degree of a minimum-degree MST over any finite pointset in $\Re_p^k$ space exceeds 2^k; that is, for all k there exists a p such that $\hat{\nu}(k,p) > 2^k$.*

Proof: Consider the pointset $(-1, \pm 1, \ldots, \pm 1)$, $(\epsilon, \pm\delta, \ldots, \pm\delta)$, and $(k^{1/p}, 0, \ldots, 0)$, where $(\epsilon^p + (k-1)\delta^p) = k$. It is possible to choose ϵ, δ, and p so that each pair of points is on the surface of a L_p ball of radius $k^{1/p}$, and all interdistances are greater than $k^{1/p}$. □

5 Conclusion

Motivated by practical VLSI applications, we showed that the maximum possible vertex degree in an L_p MST equals the Hadwiger number of the corresponding unit ball, and we determined the maximum vertex degree in a minimum-degree L_p MST. We gave an exponential lower bound on the MST number of a k-crosspolytope, and showed that the MST number for an L_p unit ball, $p > 1$, is exponential in the dimension. We concentrated on the L_1 metric in two and three dimensions due to its significance for VLSI: for example, we showed that for any finite pointset in the Manhattan plane there exists an MST with maximum degree of at most 4, and that for three-dimensional Manhattan space the maximum possible degree of a minimum-degree MST is either 13 or 14.

We solved an open problem regarding the complexity of computing bounded-degree MSTs by providing the first known polynomial-time algorithm for constructing a MST with maximum degree 4 for an arbitrary pointset in the rectilinear plane. Moreover, our techniques can be used to compute a bounded-diameter MST as efficiently as an ordinary MST. Finally, our results also enable a significant execution speedup of a number of common VLSI routing algorithms. Remaining open problems include:

1. Whether the MST number for L_1 in three dimensions is 13 or 14;
2. The complexity of computing a planar Euclidean MST with maximum degree 4;
3. Tighter bounds on the Hadwiger and MST numbers for arbitrary k and p.

References

[1] T. Barrera, J. Griffith, S. A. McKee, G. Robins, and T. Zhang, *Toward a Steiner Engine: Enhanced Serial and Parallel Implementations of the Iterated 1-Steiner Algorithm*, in Proc. Great Lakes Symp. VLSI, Kalamazoo, MI, March 1993, pp. 90–94.

[2] T. Barrera, J. Griffith, G. Robins, and T. Zhang, *Narrowing the Gap: Near-Optimal Steiner Trees in Polynomial Time*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1993, pp. 87–90.

[3] T. H. Cormen, C. E. Leiserson, and R. Rivest, *Introduction to Algorithms*, MIT Press, 1990.

[4] J. T. Croft, K. J. Falconer, and R. K. Guy, *Unsolved Problems in Geometry*, Springer-Verlag, New York, 1991.

[5] W. H. E. Day and H. Edelsbrunner, *Efficient Algorithms for Agglomerative Hierarchical Clustering Methods*, J. Classification, 1 (1984), pp. 1–24.

[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: a Guide to the Theory of NP Completeness*, W. H. Freeman, San Francisco, 1979.

[7] G. Georgakopoulos and C. H. Papadimitriou, *The 1-Steiner Tree Problem*, J. Algorithms, 8 (1987), pp. 122–130.

[8] R. L. Graham and P. Hell, *On the History of the Minimum Spanning Tree Problem*, Annals of the History of Computing, 7 (1985), pp. 43–57.

[9] L. J. Guibas and J. Stolfi, *On Computing all North-East Nearest Neighbors in the L1 Metric*, Information Processing Letters, 17 (1983), pp. 219–223.

[10] A. C. Harter, *Three-Dimensional Integrated Circuit Layout*, Cambridge University Press, New York, 1991.

[11] J.-M. Ho, G. Vijayan, and C. K. Wong, *New Algorithms for the Rectilinear Steiner Tree Problem*, IEEE Trans. Computer-Aided Design, 9 (1990), pp. 185–193.

[12] G. A. Kabatjansky and V. Levenšteĭn, *Bounds for Packings of the Sphere and in Space*, Prob. Information Trans., 14 (1978), pp. 1–17.

[13] A. B. Kahng and G. Robins, *A New Class of Iterative Steiner Tree Heuristics With Good Performance*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 893–902.

[14] M. Kruskal, *On the Shortest Spanning Subtree of a Graph, and the Traveling Salesman Problem*, Proc. Amer. Math Soc., 7 (1956), pp. 48–50.

[15] C. H. Papadimitriou and U. V. Vazirani, *On Two Geometric Problems Relating to the Traveling Salesman Problem*, J. Algorithms, 5 (1984), pp. 231–246.

[16] B. T. Preas and M. J. Lorenzetti, *Physical Design Automation of VLSI Systems*, Benjamin/Cummings, Menlo Park, CA, 1988.

[17] A. Prim, *Shortest Connecting Networks and Some Generalizations*, Bell Syst. Tech J., 36 (1957), pp. 1389–1401.

[18] J. S. Salowe and D. M. Warme, *An Exact Rectilinear Steiner Tree Algorithm*, in Proc. IEEE Intl. Conf. Computer Design, Cambridge, MA, October 1993, pp. 472–475.

[19] G. F. Tóth, *New Results in the Theory of Packing and Covering*, Convexity and its Applications, 1983.

[20] A. D. Wyner, *Capabilities of Bounded Discrepancy Decoding*, AT&T Tech. J., 44 (1965), pp. 1061–1122.

Optimal Linear-Time Algorithm for the Shortest Illuminating Line Segment in a Polygon

(Extended Abstract)

Gautam Das *†   Giri Narasimhan†

Abstract

Given a simple polygon, we present an optimal linear-time algorithm that computes the *shortest illuminating line segment*, if one exists; else it reports that none exists. This solves an intriguing open problem by improving the $O(n \log n)$-time algorithm [Ke87] for computing such a segment.

1 Introduction

In this paper we present an optimal linear-time algorithm to do the following: given a simple polygon with n vertices, it computes in $O(n)$ time the shortest illuminating line segment inside the polygon, if one exists; otherwise it reports that no such segment exists. In other words, it computes the shortest line segment within a polygon that needs to be illuminated in order for the interior of the polygon to be lit. The algorithm solves an outstanding open problem by improving the $O(n \log n)$-time algorithm presented by Ke [Ke87] for this problem.

The concept of *weak visibility* in polygons was introduced by Avis and Toussaint [AT]. Since then it has received much attention from researchers [AT, BMT, SS, TL, DHN1, DHN2, Che, DC, IK, Ke87]; also see the survey article by J. O'Rourke [O'R]. Two sets of points are *weakly visible* from each other if *every* point in either set is visible from *some* point in the other set. Thus, our algorithm computes the shortest line segment that is weakly visible from a given simple polygon.

An interesting related result due to Bhattacharya and Mukhopadhyay [BM] is a linear-time algorithm to compute a *single* weakly-visible line segment in a polygon. Although a linear-time algorithm was presented for this problem in [DHN2], the strength of their result lies in the fact that their algorithm does not use the linear-time triangulation algorithm [Cha], or the linear-time algorithm to compute shortest paths in a triangulated polygon [LP, GHLST]. The present algorithm, in contrast, uses both the triangulation as well as the shortest path algorithms, but manages to compute more; it computes the *shortest* weakly-visible segment in a polygon. Another related result is an optimal linear-time algorithm due to Bhattacharya et al. [BMT] for computing the shortest line segment from which the exterior of a simple polygon is weakly visible. It may be noted that we are concerned with internal weak visibility.

Besides resolving a long-standing open problem, our paper is also interesting because of the techniques used. The results in this paper build on some of our previous work on optimal linear-time algorithms for weak-visibility problems in polygons. The linear-time algorithms for computing *all LR-visible pairs of points* [DHN1] and for computing *all weakly-visible chords* [DHN2] output a mass of information related to visibility within a polygon. Our present algorithm shows how to exploit this wealth of information to answer more interesting questions related to weak visibility in polygons. We achieve our results by studying the structure of *minimal* weakly-visible segments and identifying the bounding chords for such segments. As described later, one of the by-products of our algorithm in this paper is a linear-time algorithm to generate all minimal weakly-visible segments. These techniques were also used in [DHN2] to obtain a linear-time recognition of L_2*-convexity* of simple polygons.

The shortest illuminating line segment in a poly-

[*] Supported in part by NSF Grant CCR-930-6822

[†] Math Sciences Dept., Memphis State University, Memphis, TN 38152. *e-mail:* dasg/giri@next1.msci.memst.edu

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

gon can also be thought of as the shortest straight line path that a watchman could patrol along in order to watch over a polygonal art gallery. There have been a number of papers on the *shortest watchman tour* problem [CN, PV]. Our algorithm finds the shortest straight-line watchman tour, if one exists.

2 Notation

We define notation for this paper. A *polygonal chain* is a concatenation of line segments. The endpoints of the segments are called *vertices*, and the segments themselves are *edges*. If the segments intersect only at the endpoints of adjacent segments, then the chain is *simple*, and if a polygonal chain is closed we call it a *polygon*. In this paper, we deal with a simple polygon P of n vertices, and its interior, $int(P)$. The segment between two points x and y is denoted $\overline{xy}$, and $int(\overline{xy}) = \overline{xy} \setminus \{x, y\}$. Two points $x, y \in P$ are *visible* (or *co-visible*) if $\overline{xy} \subset P \cup int(P)$. Two sets of points A and B are said to be *weakly visible* from each other if *every* point in A is visible from *some* point of B, and vice versa. A polygon is said to be L_2–convex if for every pair of points in the polygon, there exists another point from which the first two are visible. The (Euclidean) distance between two points x and y is denoted $dist(x, y)$. We assume that the input is in general position, which means that no three vertices are collinear, and no three lines defined by edges intersect in a common point.

If x and y are points of P, then $P_{CW}(x, y)$ ($P_{CCW}(x, y)$) is the subchain obtained by traversing P clockwise (counterclockwise) from x to y. The subchains $P_{CW}(x, y)$ and $P_{CCW}(x, y)$ includes their endpoints x and y. A polygon P is said to be *LR-visible* with respect to a pair of points x and y, if $P_{CW}(x, y)$ is weakly visible from $P_{CCW}(x, y)$.

We let $\vec{r}(x, \alpha)$ represent the ray rooted at x in direction α. For a vertex x of P, let x^+ be the vertex adjacent to x in the clockwise direction, and x^- the vertex adjacent in the counterclockwise direction. Informally, the *ray shot* from a point $x \in P$ in direction α consists of "shooting" a "bullet" from x in direction α which travels until it hits a point of P. The first point where this shot hits P is called the *hit point* of the ray shot. The line segment from a shooting point x to its hit point x' is called a *chord* of the polygon. A *weakly-visible chord* is a chord that is weakly visible from the rest of the polygon. A *weakly-visible segment* is simply any line segment in $P \cup int(P)$ that is weakly visible from the rest of the polygon.

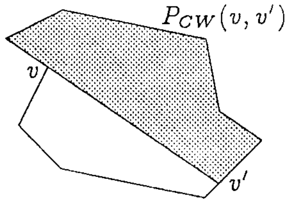


Figure 1: A clockwise component and its C-polygon

Each reflex vertex v defines two special ray shots as follows. If α is the direction from v^- to v, we let $\vec{r}_{CW}(v) = \vec{r}(v, \alpha)$ represent the *clockwise ray shot* from v. If v' is the hit point of the clockwise ray shot, then the subchain $P_{CW}(v, v')$ is the *clockwise component* of v (see Fig. 1). Counterclockwise ray shots and components are defined in the same way. A component is *redundant* if it is a superset of another component.

The clockwise component of v also defines a subpolygon called the *clockwise C-polygon* of v, which is the subpolygon of P bounded by the polygonal chain $P_{CW}(v, v')$ and the chord $\overline{vv'}$. The clockwise C-polygon of v is shown as a shaded region in Fig. 1. The counterclockwise C-polygons are defined in a similar fashion. We define the *envelope* of a C-polygon to be the boundary (in the interior of P) of the C-polygon of v. Note that the envelope of a C-polygon is simply the straight-line segment (chord) consisting of the ray shot from v. Later, we will refer to the *envelope* of an intersection of a set of C-polygons, which is the boundary (inside of P) of the intersection of the set of C-polygons.

3 Preliminaries

In this section we describe some of the geometric properties of a weakly-visible line segment.

It was noted in [IK] that the family of non-redundant components completely determines LR-visibility of P, since a pair of points s and t admits LR-visibility if and only if each non-redundant component of P contains either s or t. A similar result from [DHN2] states that the family of non-redundant components also determines all weakly-visible chords, since a chord $\overline{st}$ is a weakly-visible chord if and only if each non-redundant component of P contains either s or t. Lemma 1 below shows that the family of non-redundant components also determines the family of weakly-visible segments, while lemma 2 describes another property satisfied by all weakly-visible segments in P.

Lemma 3.1 *A line segment $\overline{uv} = l$ is weakly visible from P iff the segment l intersects the C-polygon corresponding to every non-redundant component of P.*

Lemma 3.2 *If a line segment $\overline{uv} = l$ is weakly visible from P, then the chord l', which is obtained by extending l in both directions until it hits P, is a weakly-visible chord, and the endpoints of l' form a LR-visible pair of points with respect to P.*

The obvious implication of Lemma 3.2 is that a polygon has at least one weakly-visible chord iff it has at least one weakly-visible segment and consequently a shortest weakly-visible segment.

Before giving an overview of the algorithm, we describe the peculiar output of the $O(n)$-time algorithm for computing *all weakly-visible chords* of a polygon [DHN2] (the chords algorithm, in short), since this algorithm is used by our scheme. The chords algorithm generates $k = O(n)$ pairs of the form (A_i, B_i). Note that for the rest of the paper k will be used to denote the number of pairs output by the chords algorithm. Here A_i is a line segment on P that is described in terms of a parameter x (for example, $x = 0$ (1) refers to the left (resp. right) endpoint of A_i). B_i is a polygonal chain on P such that every line segment joining a point on A_i and a point on B_i forms a weakly-visible chord. However, B_i has endpoints that are linear functions of the parameter x. It may also be noted that each of the A_is are disjoint line segments, while the B_is are possibly overlapping polygonal chains.

4 Overview of algorithm

The first step of our algorithm is to run the chords algorithm. If the polygon has no weakly-visible chords, then the algorithm stops and declares that there are no weakly-visible segments either.

Note that for the rest of the paper all references to the term components are references to non-redundant components. For every A_i output by the chords algorithm, let SA_i be the set of components that contain A_i, and let CA_i be the intersection of the C-polygons corresponding to the components in SA_i. Let the envelope of CA_i be denoted by α_i. Let SB_i be the set of all components not in SA_i, and let CB_i be the intersection of the C-polygons of all the components in SB_i. Let β_i denote the envelope of CB_i.

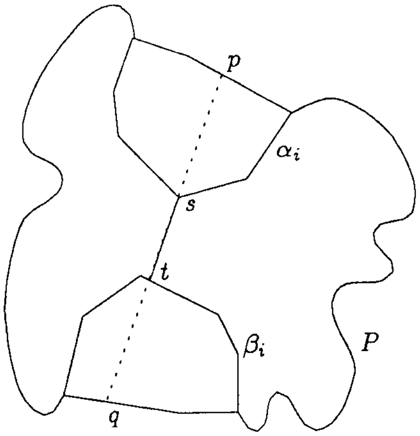


Figure 2:

It is clear that any line segment inside P that touches both α_i and β_i must intersect every C-polygon and by lemma 3.1 must be a weakly-visible segment. However, the converse is not so obvious. Lemma 4.1 below, which implies the converse, proves that the shortest weakly-visible segment must join a point on α_i and a point on β_i, for some i. This vital property is necessary to make our algorithm work in linear time. Note that SA_i corresponds to a subsequence of the sorted sequence of components. What lemma 4.1 proves is that only such *subsequences* (and not an arbitrary *subset*) of non-redundant components need to be considered for computing the shortest weakly-visible segment.

Lemma 4.1 *If $\overline{st} = l$ is the shortest weakly-visible segment, then s must lie on α_i and t must lie on β_i, for some $i = 1, \ldots, k$.*

Proof:

Extend the line segment $l = \overline{st}$ until it hits P at points p and q as shown in Fig. 2. By Lemma 3.1, segment l must intersect every C-polygon. This implies that for every C-polygon C, one of the two line segments $\overline{ps}$ and $\overline{qt}$ must lie completely in C. Also, if the envelope of C intersects l', it must include precisely one of the two segments $\overline{ps}$ and $\overline{qt}$.

By the minimality of l, it is clear that the point s (t) lies on the envelope of one or two C-polygons. If s (resp. t) lies on the envelope of some C-polygon C_s (resp. C_t), then since the envelope of C_s (resp. C_t) intersects l', C_s (resp. C_t) includes the point p (resp. q), and does not include q (resp. p).

Let $l' = \overline{pq}$. By Lemma 3.2, l' is a weakly-visible chord. Consider the output of the chords algorithm.

Let $p \in A_i$. By the correctness of the chords algorithm, $q \in B_i$. Note that C_s must include all of A_i because of the way the chords algorithm works. Thus, C_s must be a C-polygon that determines α_i, and C_t must be a C-polygon that determines β_i. Hence s must lie on α_i and t must lie on β_i thus concluding the proof. □

The above lemma suggests the following skeleton for our algorithm. For every $i = 1, \ldots, k$, construct the envelopes α_i and β_i, and then compute the shortest line segment joining a point on α_i and a point on β_i. The shortest of these is the shortest weakly-visible segment.

Note that since both α_i and β_i are convex polygonal chains, computing the shortest line segment connecting them can be computed in time $O(|\alpha_i| + |\beta_i|)$, where $|\alpha_i|$ and $|\beta_i|$ are the lengths of the two chains. However, in general there may be considerable overlap between α_i and α_{i+1}, as well as between β_i and β_{i+1}. For the i-th iteration, instead of simply finding the shortest line segment that joins α_i and β_i, the algorithm only scans the portion of α_i that is not part of α_{i+1} (and the portion of β_i that is not part of β_{i+1}), and finds the shortest segment between those parts. The assumption is that the rest of the portions of the two polygonal chains will be scanned as part of a later iteration. Repetitious scanning of the polygonal chains is thus prevented by delaying the scanning of overlapping portions as much as possible.

In section 5.1, we precisely characterize how α_i changes to become α_{i+1}, and correspondingly how β_i changes to become β_{i+1}. In section 5.2, we describe a data structure that stores $\alpha_i, i = 1, \ldots, k$, and another identical structure that stores $\beta_i, i = 1, \ldots, k$. The planarity of these data structures is sufficient proof that the size of the union of α_i and the size of the union of β_i for $i = 1, \ldots, k$ is $O(n)$.

The problem with the skeleton algorithm described above is that the shortest line segment joining α_i and β_i for some i may not lie entirely within P. This happens because even though the line segment when extended may hit A_i, it might not hit B_i because of obstruction from the rest of P, i.e., the extended line is not a weakly-visible chord. In this case, if there is a weakly-visible chord connecting A_i and B_i, then the shortest weakly-visible segment joining α_i and β_i would touch a vertex of the P. This suggests that our algorithm needs to deal with two main cases. The first case is when the shortest illuminating segment does not touch a vertex of P except at its endpoints; the second case is when it touches a vertex of P in its interior. If the first case occurs, the algorithm briefly described earlier will output the shortest illuminating segment. The details of this case are described in the next section. The second case is handled separately in section 6. The algorithm for the second case is a modification of our earlier algorithm for computing all weakly-visible chords of a polygon [DHN2]. If a weakly-visible segment touches a vertex of P it is referred to as a *non-tangential weakly-visible segment*; otherwise it is referred to as a *tangential weakly-visible segment.*

By putting all the pieces together, we show a linear-time algorithm to obtain the shortest non-tangential weakly-visible segment, and a linear-time algorithm to compute the shortest tangential weakly-visible segment. The shortest of the two segments is the shortest weakly-visible segment in a polygon, thus giving us the desired algorithm.

5 Case 1: Non-tangential weakly-visible segment

As mentioned earlier, this case corresponds to the situation when the shortest weakly-visible segment does not touch any vertex of the polygon except possibly at its endpoints. For each $i = 1, \ldots, k$, let SN_i be the shortest non-tangential weakly-visible segment that joins α_i and β_i with at least one endpoint on $\alpha_i - \alpha_{i+1}$ or $\beta_i - \beta_{i+1}$. It can be shown that the shortest of the segments SN_i, $i = 1, \ldots, k$ must be the shortest non-tangential weakly-visible segment that joins α_i and β_i, $i = 1, \ldots, k$.

5.1 Structure of α_i and β_i

As mentioned earlier both α_i and β_i are the boundaries (internal to P) of the intersection of a set of C-polygons. Hence it is clear that both of them are convex polygonal chains. It may be possible that $\alpha_i = \alpha_{i_1} = \alpha_{i_2} = \cdots = \alpha_{i_p}$. This simply means that no component starts or ends on the portion of P covered by $A_{i_1}, A_{i_2}, \ldots, A_{i_p}$.

We now describe the structural differences between α_i and α_{i+1} (in case they do differ), and the corresponding differences between β_i and β_{i+1}. The main purpose of studying this structure is to identify the polygonal chains $\alpha_i - \alpha_{i+1}$ and $\beta_i - \beta_{i+1}$ so that they can be processed in the i-th iteration. Clearly, if $\alpha_i = \alpha_{i+1}$, then no processing is required in iteration i.

Assume that $\alpha_i \neq \alpha_{i+1}$. From [DHN2] we know that A_i and A_{i+1} are disjoint line segments. There are various events that trigger the chords algorithm to go from iteration i to iteration $i+1$, thus outputting pairs (A_i, B_i) and pairs (A_{i+1}, B_{i+1}). An event occurs if a component starts or ends between A_i or A_{i+1}. The other possible events have to do with changes in the points of tangency for the boundaries of the weakly-visible chords. The chain α_i is different from α_{i+1} only when a component starts or ends between A_i and A_{i+1}. For the next three paragraphs we will assume that the counterclockwise end for any polygonal chain is the *front* end, while the clockwise end is the *tail* end.

If a component c starts between A_i and A_{i+1}, the changes from α_i to α_{i+1} are as shown in Fig. 3(b). Note that the C-polygon corresponding to component c lies to the left of the line and that the component c consists of the polygonal chain $P_{CW}(p_2, q_2)$. A_i lies to the right of p_2, while A_{i+1} lies to the left of p_2. α_i consists of the chain from a_t to s_2 to a_f, while α_{i+1} consists of the chain from p_2 to s_2 to a_f, i.e., a portion of the tail of α_i gets replaced by a portion of the ray shot corresponding to the component c. At the same time, as shown in Fig. 3(b) β_i has a portion of its front replaced by a new polygonal chain. β_i consists of the chain from b_t to t_2 to q_2, while β_{i+1} consists of the chain from b_t to t_2 to b_f. In other words, CA_i shrinks at its tail end, and CB_i grows at its front end, while both their boundaries remain convex. Note that $\alpha_i - \alpha_{i+1}$ comprises of the polygonal chain from a_t to s_2, while $\beta_i - \beta_{i+1}$ comprises of the segment from q_2 to t_2.

Note that in the Figs. 3(a) and (b), the region $CA_i \cap CA_{i+1}$ (as well as the region $CB_i \cap CB_{i+1}$) have been shown as a filled region. The area occupied by CA_{i+1} (but not by CA_i) is indicated as a dot-filled region, while the area occupied by CA_i and CB_i is left blank.

By a similar argument, if a component c ends between A_i and A_{i+1}, the portion of the ray shot corresponding to c at the front (right end or the counterclockwise end) of α_i gets replaced by a new polygonal chain, causing CA_i to grow in the front. As shown in figure 2(a), β_i has a portion of its tail (right end or counterclockwise end) replaced by a portion of the ray shot corresponding to c, thus causing CB_i to shrink at its tail end. In this case note that $\alpha_i - \alpha_{i+1}$ comprises of the segment from p_1 to s_1, while $\beta_i - \beta_{i+1}$ comprises of the chain from b_t to t_1.

The above description describes the changes that take place to the α and β chains while moving from the i-th iteration to the $(i+1)$-st iteration.

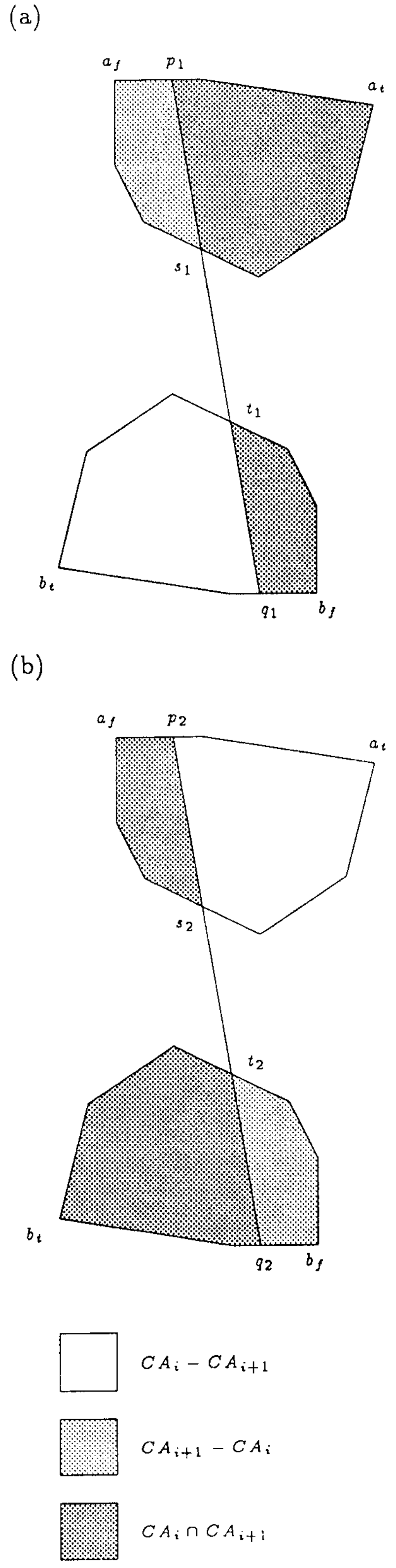


Figure 3: Changes in the structure of α_i and β_i

5.2 Data structure for storing the α and β chains

The union of all the chains α_i (β_i) form a tree. We claim here without elaborating that both the sets of chains α_i and β_i can be stored in simple linear-sized data structures that are similar to the *persistent search trees* of Sarnak and Tarjan [ST]. The data structure will be detailed in a full version of this paper.

5.3 Computing SN_i

As described in the previous section, the algorithm goes through k iterations. In the i-th iteration, the chains $\alpha_i - \alpha_{i+1}$ and $\beta_i - \beta_{i+1}$ are identified, and the shortest segment that joins α_i and β_i with one endpoint on $\alpha_i - \alpha_{i+1}$ or $\beta_i - \beta_{i+1}$ is computed. Let this segment be $SN_i = \overline{s_i t_i}$.

Given any two convex polygonal chains α and β, there is a simple sweep algorithm to find the shortest line segment that joins the two chains. In this case, α and β are two convex chains that form part of the boundary of two disjoint convex polygons. The algorithm involves sweeping the two chains, one from its clockwise end and in counterclockwise order, the other from its counterclockwise end in clockwise order. Informally speaking, the sweep algorithm works because of three simple facts: (1) for a fixed point $a \in \alpha$, its distance to visible points $b \in \beta$ is unimodal, (2) as point a moves monotonically on α, its closest point on β moves monotonically on β, (3) for points $a \in \alpha$, its shortest distance to β (i.e., the distance to its closest point on β) is unimodal. Intuitively speaking, fact (3) states that the *local minimum* is also the *global minimum.*

In the i-th iteration, it is possible that the segment $\overline{s_i t_i}$ may not lie entirely within P. To identify this situation, we exploit the fact that given a point x on P, the chords algorithm has already identified which directions from x give rise to weakly-visible chords. Hence to check whether SN_i lies in P, the algorithm computes the endpoint of the chord generated when the line segment is extended. Call this point p_i. Using the output of the chords algorithm our algorithm checks whether the chord in the direction $\overline{p_i s_i}$ is a weakly-visible chord. It should be pointed out that it is possible that p_i may not lie on A_i, but on some other segment A_j. To identify A_j, the algorithm traverses from A_i to A_j along P. It is non-trivial to show that this portion of P is not traversed again for this purpose (and hence does not contradict the claimed $O(n)$ time complexity). The intuition behind the claim is that if p_i lies on A_j then SN_i is also the shortest segment between α_j and β_j as well as between α_l and β_l for all values of l between i and j. On the other hand, if the chord is not weakly visible, then the segment $\overline{s_i t_i}$ is ignored, and will be handled by the second phase of the algorithm (corresponding to Case 2).

Another subtle complication is introduced by the possibility that SN_i may have one endpoint on $\beta_i - \beta_{i+1}$ and another endpoint on $\alpha_i \cap \alpha_{i+1}$ (instead of $\alpha_i - \alpha_{i+1}$). This could happen if the sweep algorithm (described at the start of this subsection) for finding the shortest line segment joining two convex polygonal chains reaches the end of one of the chains without hitting a *local minimum.* For example, assume that the end of $\alpha_i - \alpha_{i+1}$ is reached before reaching the end of $\beta_i - \beta_{i+1}$ and before a minimum was encountered. In this case, our algorithm continues sweeping on $\beta_i - \beta_{i+1}$, while continuing the sweep on $\alpha_i \cap \alpha_{i+1}$. Our algorithm needs to be modified to ensure that this portion of $\alpha_i \cap \alpha_{i+1}$ is not swept again in iteration $i+1$ (or later). In this case, it can be shown that SN_j cannot have an endpoint on this portion of $\alpha_i \cap \alpha_{i+1}$ and hence need not be considered in any later iteration. The relevant portion of $\alpha_i \cap \alpha_{i+1}$ is marked *visited* so that a sweep in a later iteration can skip over this portion of the chain. The entire arguments in this paragraph could have been carried out with α replaced by β and vice versa.

Once a local minimum is found for the i-th iteration, the algorithm also verifies if it is a global minimum for the shortest segment between α_i and β_i. If the point on $\alpha_i - \alpha_{i+1}$ or $\beta_i - \beta_{i+1}$ is not the endpoint of that subchain, then the global minimum for the shortest segment between α_i and β_i must have been reached. Otherwise, a simple test can check whether the global minimum has been reached or not. If it is not a global minimum, then SN_i can be ignored since the shortest segment between α_i and β_i connects points that are not on $\alpha_i - \alpha_{i+1}$ as well as $\beta_i - \beta_{i+1}$. Since such a segment would connect α_{i+1} and β_{i+1} it will be encountered in a later iteration. The algorithm with the minor modifications mentioned above is guaranteed to sweep every portion of the α and β chains exactly once and hence achieves the claimed linear-time complexity.

6 Case 2: Shortest tangential weakly-visible segment

This case occurs when the interior of the shortest weakly-visible segment in the polygon touches a vertex of the polygon. However, in this case, the corresponding weakly-visible chord obtained by extending the segment is also a tangential chord, i.e., it touches a vertex of the polygon in its interior. The crucial point to observe is that these are exactly the weakly-visible chords that are output by the linear-time chords algorithm [DHN2]. A suitable modification of the chords algorithm can output all tangential weakly-visible segments, of which the shortest can be computed.

The chords algorithm uses the following strategy. It traverses along the polygon in a counterclockwise direction with a point x. When x is on A_i, the points $y(x)$ and $z(x)$ corresponding to the other endpoints of the two tangential chords from x are computed. The points $y(x)$ and $z(x)$ move monotonically on P; so do the points of tangency for the tangential chords, namely $s(x)$ and $t(x)$. As x moves on A_i, there are several possible events that can take place, which would change the description of the tangents: the point $y(x)$ (or $z(x)$) could move to a vertex of P; the point $s(x)$ (or $t(x)$) could move to a vertex of P. These events cause a recomputation of the equations of the tangential chords as a function of x. In [DHN2] we show that the number of these events are $O(n)$, thus giving us a linear-time algorithm.

The modification for computing the tangential weakly-visible segments is as follows. During iteration i, the chains α_i and β_i are computed. When the point x is on A_i, the points of intersection of the tangential chords with α_i and β_i are also maintained (call them $a_1(x), a_2(x), b_1(x), b_2(x)$). The segment from $a_1(x)$ to $b_1(x)$ and the segment from $a_2(x)$ to $b_2(x)$ are the two tangential weakly-visible segments with respect to x. The situation is described in Fig. 4. There are, however, an additional number of events that could cause a change in the description of the tangential weakly-visible segments: the points $a_1(x)$ or $a_2(x)$ ($b_1(x)$ or $b_2(x)$) could move to a vertex of α_i (β_i). This would cause additional recomputations of the equations as well as the lengths of the tangential segments. The crucial point is that in between events, the length of the tangential segments can be computed in terms of x, from which the minimum can be computed for that interval in constant time. It can also be shown that the total number of events that will be encountered is $O(n)$ and that $a_1(x), b_1(x), a_2(x), b_2(x)$ move monotonically on the α and β chains.

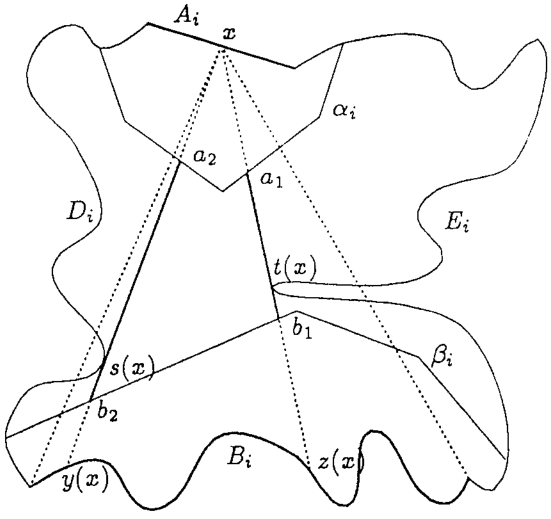


Figure 4: Case 2: Determining tangential shortest weakly-visible segments

7 All minimal weakly-visible segments algorithm

One of the by-products of our algorithm is a linear-time algorithm to generate all minimal weakly-visible segments of a polygon. This algorithm is a modification of the algorithm described in section 6 for computing the shortest tangential weakly-visible segment. It outputs a set of pairs $(U_i, V_i), i = 1, \ldots, m$. Here U_i and V_i are subchains of the polygonal chains α and β, $m = O(n)$, and any segment joining points $u \in U_i$ and $v \in V_i$ is a minimal weakly-visible segment. One note of caution is that U_i and V_i have left and right endpoints that are linear functions of a parameter x in a spirit similar to that of the endpoints of the chain B_i that is output by the chords algorithm. For a point x on A_i, the polygonal chains α_i and β_i can be computed along with the points $a_1(x), a_2(x) \in \alpha_i$ and $b_1(x), b_2(x) \in \beta_i$. The output of the algorithm consists of $(U_i, V_i) = ((a_1(x), a_2(x)), (b_2(x), b_1(x)))$. The discussion at the end of section 6 can also be used to show that the number of these pairs produced is $m = O(n)$. Lemma 1 can be used to show that these segments are minimal in the sense that any subsegment of these segments is not weakly visible.

8 Conclusion and open problems

We show optimal linear-time algorithms to compute the shortest weakly-visible segment and all minimal weakly-visible segments in a given simple polygon. Some interesting open questions are:

- Can the *exhaustive* sweeping techniques from this paper be used to solve other weak visibility problems efficiently? For example, are there linear-time algorithms for the *all-pairs* version of any of the 2-guard walk problems (see [DHN1])?
- Can the algorithm from this paper be designed without using the triangulation or the shortest path algorithm as a subroutine?
- Ntafos [Nt] introduced the notion of *d–visibility*, where an observer's visibility is limited to distance d. Can the shortest illuminating segment be computed efficiently under d–visibility?

References

[AT] D. Avis and G.T. Toussaint, "An optimal algorithm for determining the visibility of a polygon from an edge," *IEEE Transactions on Computers*, **30** (1981), pp. 910-914.

[BMT] B. K. Bhattacharya, A. Mukhopadhyay, and G.T. Toussaint, "A linear-time algorithm for computing the shortest line segment from which a polygon is weakly externally visible," in *Proc. of 2nd WADS*, 1991, pp. 412-424.

[BM] B. K. Bhattacharya, and A. Mukhopadhyay, "Computing in linear time an internal line segment from which a simple polygon is weakly internally visible," Personal Communication, 1993.

[Cha] B. Chazelle, "Triangulating a simple polygon in linear time," *Discrete and Computational Geometry*, **6** (1991), pp. 485-524.

[GHLST] L. Guibas, J. Hershberger, D. Leven, M. Sharir and R. Tarjan, "Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons," *Algorithmica*, **2** (1987), pp. 209-233.

[Che] D.Z. Chen, "Optimally computing the shortest weakly visible subedge of a simple polygon," Tech. Report No. 92-028, Dept. of Computer Sciences, Purdue University, May 1992.

[CN] W.P. Chin, and S. Ntafos, "Shortest Watchman Routes in Simple Polygons," *Discrete and Computational Geometry*, **6** (1991), pp. 9-31.

[DHN1] G. Das, P. Heffernan and G. Narasimhan, "LR-visibility in polygons,", *Proc. 5th Canadian Conference on Computational Geometry*, (1993), pp. 303-308.

[DHN2] G. Das, P. Heffernan and G. Narasimhan, "Finding all weakly-visible chords of a polygon in linear time," manuscript, 1993.

[DC] J. Doh and K. Chwa, "An algorithm for determining internal line visibility of a simple polygon," *J. of Algorithms*, **14** (1993), pp. 139-168.

[H] P. J. Heffernan, "An optimal algorithm for the two-guard problem," *Proc. Ninth Annual ACM Symp. on Computational Geometry*, 1993, pp. 348-358; to appear in *Inter. J. on Computational Geometry and Applications.*

[IK] C. Icking and R. Klein, "The two guards problem," *Inter. J. on Computational Geometry and Applications*, **2** (1992), pp. 257-285.

[Ke87] Y. Ke, "Detecting the weak visibility of a simple polygon and related problems," Tech. Report, The Johns Hopkins University, (1987).

[LP] D. T. Lee and F. P. Preparata. "An optimal algorithm for finding the kernel of a polygon," *JACM*, **26**(3), (1979), pp. 415-421.

[Nt] S. Ntafos, "Watchman Routes Under Limited Visibility," *Computational Geometry: Theory and Applications*, **1**(3) (1991), pp. 149-170.

[O'R] J. O'Rourke, "Computational geometry column 18," *SIGACT News*, **24** (1993), pp. 20-25.

[PV] P. Pradeep Kumar, and C.E.Veni Madhavan, "Shortest watchman tours in weak visibility polygons,", *Proc. 5th Canadian Conference on Computational Geometry*, (1993), pp. 91-96.

[SS] J.-R. Sack and S. Suri, "An optimal algorithm for detecting weak visibility," *IEEE Transactions on Computers*, **39** (1990), pp. 1213-1219.

[ST] Sarnak and R. Tarjan, "Planar Point Location Using Persistent Search Trees", *CACM*, **29**, (1986).

[TL] L.H. Tseng and D.T. Lee, "Two-guard walkability of simple polygons," manuscript, 1993.

Morphing Simple Polygons

Leonidas Guibas
Stanford University
Stanford, CA 94305

John Hershberger[*]
Mentor Graphics
San Jose, CA 95131

Abstract

In this paper we investigate the problem of morphing (i.e. continuously deforming) one simple polygon into another. We assume that our two initial polygons have the same number of sides n, and that corresponding sides are parallel. We show that a morph is always possible by a varying simple interpolating polygon also of n sides parallel to those of the two original ones. If we consider a uniform scaling or translation of part of the polygon as an atomic morphing step, then we show that $O(n^{4/3+\epsilon})$ such steps are sufficient for the morph.

1 Introduction

In computer graphics a recent area of interest has been *morphing*, i.e. continuously transforming one shape into another. Morphing algorithms have numerous uses in shape interpolation, animation, video compression, as well as obvious entertainment value. Some of the relevant graphics papers are [KR, SG, SGWM]. In general there are numerous ways to interpolate between two shapes and little has been said about criteria for comparing the quality of different morphs and notions of optimality. A common problem with many morphing algorithms is that although locally the geometry of the interpolated shape remains faithful to the original shapes, global properties such as symmetry, simplicity (in the topological sense), etc., can be easily violated. In fact, the very existence of morphs that can maintain high-level global properties of the interpolated shapes is often far from obvious.

In this paper we address a morphing problem raised by John F. Hughes of Brown University in the Pixels-and-Predicates Workshop organized by David Salesin and Jack Snoeyink at the University of British Columbia last spring. The problem is set in two dimensions; we are given two simple polygons P and Q. Assume that both P and Q are counterclockwise oriented. The polygons P and Q are *parallel*, where this is taken to mean that P and Q have the same number of sides and their sides can be put in one to one correspondence so that corresponding sides are parallel and oriented the same way. This is equivalent to saying that that the two polygons have the same sequence of angles. The morphing problem is to interpolate between P and Q via a continuously changing polygon $R(t)$, where t denotes time. We set $R(0) = P$ and $R(1) = Q$. We require that $R(t)$ remain parallel to P and Q at all times, and that it also remain simple. See Figure 1 below.

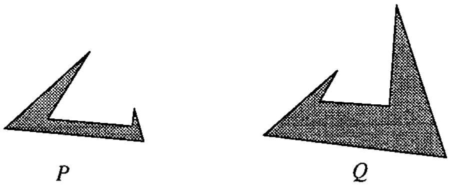


Figure 1: Two parallel polygons

We provide a discrete solution to Hughes's problem. We take as our basic operation the uniform scaling and/or translation of a contiguous part of a polygon parallel to itself. We call such a step a *parallel move* of the corresponding polygonal chain. The affected edges in a parallel move expand or contract so as to keep the two endpoints of the chain on the lines supporting the two adjacent edges to the chain. Also, no parallel move is allowed that would cause any moving edge or vertex to pass over another vertex or edge of the polygon. It is clear that any number of parallel moves keep a simple polygon parallel to itself and simple. Our main result is that, if n denotes the number of sides of P and Q, then $O(n^{4/3+\epsilon})$ parallel moves always suffice to morph P to Q. We have an algorithm of the same complexity that

[*]Most of this research was performed while the second author was visiting at Stanford University.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

computes such a sequence of moves.

The problem of constructing a simple polygon given a sequence of angles has a substantial history in computational geometry. Culberson and Rawlins [CR] first raised this question and gave an algorithm that constructs such a polygon in time $O(nD)$, where the assumption is that all angles are rational multiples of π by a fraction whose denominator is bounded by D. Vijayan and Wigderson improved this result by constructing such a polygon in linear time with no restriction on the angles being rational (reported in Vijayan [V]). The result of the present paper paper shows that all simple polygons realizing the same sequence of angles are homotopic to each other within this class of polygons. This in itself is an interesting result which, according to Hughes, had been in the folklore but never formally proven. Some of the trickiness involved can be seen in the two polygons of Figure 2. These are oppositely spiraling polygons, but they are still parallel and therefore morphable.

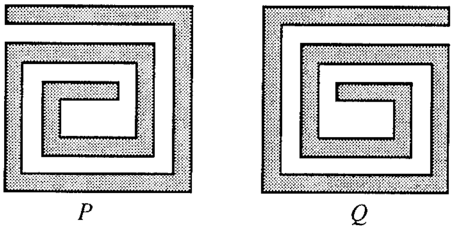


Figure 2: These oppositely spiraling polygons are still morphable

The main idea of this paper is to use parallel moves to transform each polygon to a reduced form in which the side lengths are of widely different scales. Once both polygons are in reduced form, we can use a sequence of elementary local transformations to morph one reduced form to the other. In the reduced form, small-scale structures do not interfere with larger-scale parallel moves—when a large-scale edge translates, any smaller-scale structures adjacent to its endpoints translate along as if they were part of the endpoints. We establish a correspondence between these transformations in reduced polygons and (classical) rotations in binary trees. The catch, however, is that our trees have weights on the leaves and induced weights on the internal nodes. These weights correspond roughly to the polygon angles. Our mapping from polygons to trees is such that all weights of leaves or internal nodes must be in the range $(-1, +1)$. Although the problem of using single rotations to transform one binary tree to another is well-studied and is known to be doable in a linear number of rotations, these methods are not applicable to our situation, as not all rotations maintain the weight condition on the trees. The combinatorial essence of our morphing algorithm becomes then a theorem about rotating one weighted binary tree into another while satisfying the weight conditions. This in itself is quite an interesting combinatorial problem.

We note that for rectilinear polygons the problem is easier, and a solution is already known [T]. We also remark that our transformation to a reduced form is unattractive from a practical point of view, as it may change wildly the scale of different features of the original polygons. We discuss at the end of the paper some concrete and interesting algorithmic issues that arise in keeping small the overall distortion of the original polygons during the morph process.

2 The reduced form of a polygon

Our morphing algorithm works by transforming each of the two polygons P and Q into a reduced form. We first transform P to its reduced form, then morph the reduced form of P to that of Q, and then undo the transformations that reduced Q. We will be able to map the morphing of the reduced forms to the purely combinatorial tree rotating problem mentioned above.

The *reduced form* of polygon P is a polygon parallel to P with a particular hierarchical structure. The reduced form of P can be obtained from P by $O(n)$ elementary parallel moves. The reduced form is not unique, but instead depends on the order in which the parallel moves are performed.

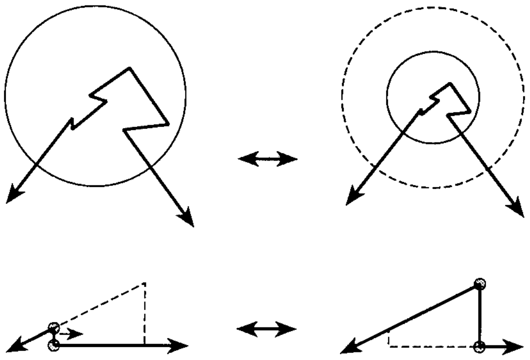


Figure 3: The two kinds of parallel moves

We allow two types of parallel moves, defined as follows: (1) Suppose there exists a circle that intersects exactly two edges of the polygon. Then we may shrink or expand the structure inside the circle uniformly, so long as the intersections of the polygon with the circle do not change. (2) Suppose there exist two disjoint circles, each intersecting an edge e of the polygon and one other edge. Then we may translate e and move the two circles and their contents uniformly along with e, such that each circle's intersections with the polygon are stationary in the reference frame of the circle. In both cases we enforce the condition that the polygon always intersects each circle in exactly two fixed places. See Figure 3.

The reduced form of P is based on a process of eliminating P's edges one by one. Consider an edge e of polygon P whose endpoints u and v are shared with adjacent edges e_u and e_v. If we translate e parallel to itself while keeping the rest of the polygon fixed, u and v move along the lines supporting e_u and e_v, lengthening or shortening those edges as well as e itself. We can translate e so that at least one of e, e_u, and e_v gets shorter. One of the edges will eventually be reduced to zero length, if e does not hit another part of P first. See Figure 4.

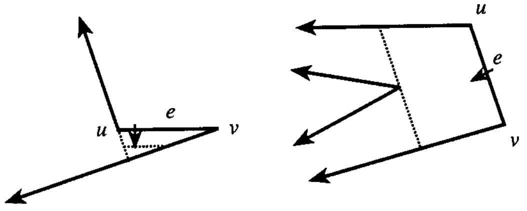


Figure 4: Translating an edge

We show in Lemma 1 below that every polygon can be morphed with $O(1)$ edge translations (amortized) so that at least one edge is reduced to zero length, while keeping the polygon simple. In this case we can iteratively reduce the polygon to a triangle: at each of $n-3$ steps, we eliminate one edge of P by translation. For certain degenerate polygons, it is possible that the elimination process will terminate with a parallelogram instead of a triangle. This case does not pose any additional difficulties for our algorithm, and so we will ignore this possibility from here on.

The reduced form of a polygon models the edge elimination process in its structure. To compute it, we run the edge elimination process as above, except that instead of reducing edges to zero length, we reduce them to infinitesimal length. Suppose that the edge elimination process eliminates an edge e, coalescing its vertices u and v to become the common endpoint of adjacent edges e_u and e_v. In the reduced form of the polygon, vertices u and v are kept separate, joined by an infinitesimal edge e. We imagine drawing an infinitesimal disk that contains u and v, centered on the intersection of the lines supporting e_u and e_v. (If the supporting lines are parallel, we drop the centering requirement.) In the remainder of the elimination process, the disk translates along with the intersection of the supporting lines, acting like a finite-size vertex. Because the disk and its contents are infinitesimal, we are sure that no translating edge touches the disk unless it also touches the corresponding vertex in the original edge elimination process. The infinitesimal edge is hidden inside the disk and does not participate directly in the remainder of the elimination process. See Figure 5.

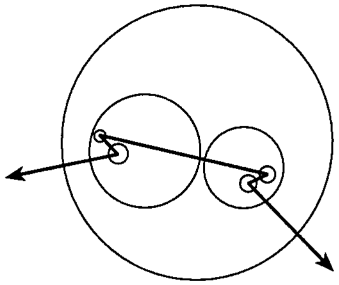


Figure 5: A reduced form

When the edge elimination process terminates, the reduced form of the polygon consists of a triangle with some *microstructure* at its vertices. Each vertex is contained in a disk, and the three disks are disjoint. If we look inside a disk, we see two *infinite* edges—the ones that cross the disk boundary—and a single *finite* edge joining them. The vertices where the finite edge joins the infinite edges have microstructure themselves: they are contained inside smaller disks that are disjoint from each other and completely inside the surrounding disk; each smaller disk contains recursively similar structures.

If we ignore the disks of the reduced form and look only at the edges, we see that the reduced form of a polygon P is a special hierarchical morph of P: the reduced form is parallel to P, and it is obtained from P by parallel moves that preserve simplicity.

The following lemma, due to Emo Welzl, shows that the edge elimination process applies to any polygon. Combining this with the discussion above, we can compute a reduced form for any polygon.

Lemma 1 ([W]). *Given a simple polygon P with n vertices, we can reduce it to a triangle by a sequence of $O(n)$ edge translations, each of which preserves simplicity and $n-3$ of which shorten some edge to zero length.*

Proof: The algorithm repeatedly translates edges toward the interior of the polygon. It maintains a set of *frozen* edges that have already been translated and may not be translated again. It works in a subpolygon of P, and maintains the invariant that at most two (adjacent) edges of the current subpolygon are frozen.

The generic step of the algorithm picks an arbitrary unfrozen edge e of the current polygon (initially P) and translates it toward the interior of P. As the edge translates parallel to itself, its endpoints move along the lines supporting the two edges adjacent to e, and those two edges shorten or lengthen. The translation stops when one of two things happens: (1) e or one of its two neighboring edges is shortened to zero length, or (2) e collides with some vertex v of P, or a translating endpoint v of e collides with some other edge of P. See Figure 6a.

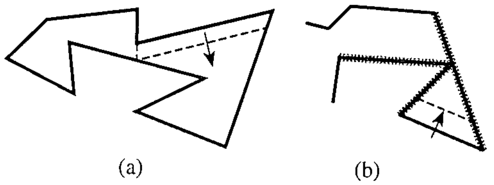


Figure 6: Eliminating edges to find a reduced form

In case (1) we have eliminated an edge. In case (2), the collision between an edge and the vertex v has partitioned the polygon in two, and we recursively apply the same procedure on one of the two pieces. Specifically, we translate e away from the polygon interior slightly, so the polygon is not completely partitioned by the collision at v (the amount to back away is upper-bounded by a constant that can be computed from the geometry of the polygon), and freeze all three edges participating in the collision: the two edges incident to v, and the edge v hit. Because at most two adjacent edges of the current polygon were frozen, the same is true of at least one of the two subpolygons created by the near-partition at v. We recurse on the subpolygon with fewer frozen edges. Each of the subpolygons is at least a triangle, so each also has fewer edges than the current polygon.

The recursion bottoms out, in the worst case, when the current polygon is a triangle with two frozen edges (Figure 6b). Translating the third edge inward eliminates one of the frozen edges. When a frozen edge is eliminated, it contracts into the vertex v that caused its freezing. We unfreeze the other two edges that were frozen because of v. This restores the invariant: the collision at v originally created two subpolygons P_1 and P_2, each with at least three edges, with one edge belonging to both P_1 and P_2. The edge elimination reduces the total number of edges in P_1 and P_2 by one, to a minimum of four. The new subpolygon is the union of P_1 and P_2; it has at least four edges, at most two of which are frozen.

The size of the current subpolygon decreases at each recursive call, and the algorithm eliminates an edge when the current subpolygon is a triangle. Thus the algorithm successfully eliminates all the edges of P (or reduces P to a triangle, if we stop just before the last elimination). Edges are unfrozen only when we move up one recursive level. Each unfreezing can be charged to the edge elimination that occurred just before it. The total number of edge translations is bounded by n plus the number of unfreezings (at most $2n$) and hence is $O(n)$. □

The preceding algorithm is less than ideal, because it uses more than $n-3$ edge translations to reduce P to a triangle. We conjecture that there is always some edge that can be translated to eliminate itself or one of its neighbors without violating simplicity. This would give the ideal algorithm we want, but we have so far been unable to prove the conjecture.

The key feature of the reduced form is the hierarchy of edge eliminations, as the following lemma shows.

Lemma 2. *If two reduced forms are parallel polygons and have the same hierarchical structure, then one can be morphed to the other using a linear number of edge translations.*

Proof: Let P and Q be the two reduced forms. The algorithm follows the hierarchy of the reduced form. We repeatedly translate edges of P to be collinear with the corresponding edges of Q. When we are done, the two polygons are identical. The first step translates the top-level triangle edges of P to be collinear with those of Q, moving the microstructure disks along with the intersections of the edges. Within each microstructure disk, we translate the finite edge of P to be collinear with the corresponding edge of Q. (The subdisks translate along with the intersection points of the finite and infinite edges.) We then recursively make the two polygons identical inside each subdisk. □

3 The tree representation of a reduced form

The hierarchical nature of the reduced form can be naturally represented by a tree structure. The infinitesimal disks of the reduced form correspond to nodes in a binary tree; a disk and the two subdisks it contains correspond to a node and its children. Each of the three top-level disks of the reduced form corresponds to a tree. Logically these three trees should be joined in a ring; however, to maintain the convenience of a consistent binary tree representation, we omit the ring edges and work with the trees separately. Each tree represents the reduced form of a simple bi-infinite chain (one whose initial and final edges are rays).

In this section and Section 4 we work only with chains and their corresponding trees; we show how to morph two parallel chains to each other. In Section 5 we address the original problem of morphing polygons.

Each node of a tree has an angular weight associated with it, namely the turn angle at the corresponding disk, i.e., the difference between the direction angles of the incoming and outgoing edges of the disk (we assume the polygon is oriented counterclockwise). For convenience we normalize the weight of each vertex of P by dividing it by π, so the weight is in the range $(-1,+1)$; the sum of the weights over all the vertices is therefore 2. The weight at each disk of the reduced form is the sum of the weights of all the polygon vertices inside it.

Because the edges incident to each vertex of P and each disk of the reduced form make an angle strictly between $-\pi$ and π, the weight of each node in the binary tree is in the open range $(-1, +1)$. We say that a node whose weight is in this range is *valid.* If all the weights in a tree are valid, we say the whole tree is valid. The binary tree associated with the reduced form of a bi-infinite chain is always valid. The following lemma shows that the converse is also true.

Lemma 3. *Any valid tree corresponds to the reduced form of some bi-infinite chain.*

Proof: The proof is by recursive expansion. Let r be the root of the tree. The weight of r corresponds to the total turn of the chain. We draw the infinite chain edges and draw a circle $\mathcal{C}$ centered on the intersection of their supporting lines. Let α and β be the weights of the children of r. Within the circle we draw an edge that makes angles of α and β with the two infinite edges, and two smaller circles inside $\mathcal{C}$ centered on the endpoints of the new edge. Applying this process recursively with each subtree and its corresponding circle produces a reduced form for the tree. (At each step the drawing inside the innermost circle is provisional, to be replaced when the angle inside the circle is expanded.) □

In the remainder of this section we reduce the problem of morphing between two parallel bi-infinite chains to a problem of transforming one valid tree into another by rotations. Because we want the tree transformation to correspond to polygon morphing, we insist that the tree be valid at all times. In particular, the transformation may use only *valid rotations*, ones whose initial and final states are both valid. For example, in Figure 7, if A, B, and C represent node weights, it must be the case that each of A, B, C, $A+B$, $B+C$, and $A+B+C$ lies in the range $(-1, 1)$.

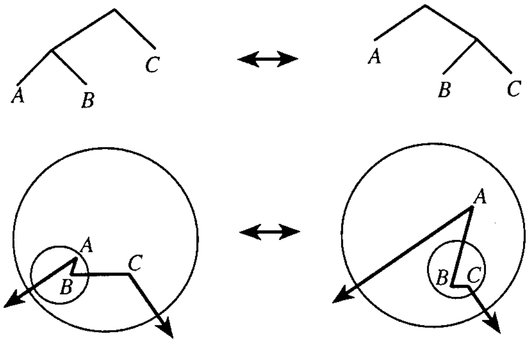


Figure 7: A tree rotation and the corresponding morph of a reduced form

Let T_1 and T_2 be two valid trees with the same sequence of weights at the leaves. Section 4 shows how to transform T_1 to T_2 using only valid rotations. We now show how to interpret a tree rotation as a morphing operation that transforms one reduced form into another. Let A, B, and C be the tree nodes involved in the rotation, grouped $((A\ B)\ C)$ before and $(A\ (B\ C))$ after the rotation. The tree structure $((A\ B)\ C)$ corresponds to a disk of the reduced form in which the edges before A and after C are infinite, the edge $\overline{BC}$ is finite, and the edge $\overline{AB}$ is infinitesimal, contained in a subdisk. Of course, A, B, and C are vertices only at this level of the reduced form; they may have microstructure of their own. See Figure 7. We want to morph $\overline{AB}$, $\overline{BC}$, and the two infinite edges such that $\overline{AB}$ becomes finite and $\overline{BC}$ infinitesimal.

Lemma 4. *Let A, B, and C be the turn angles of a four-edge bi-infinite chain, and suppose the angles $A+B$, $B+C$, and $A+B+C$ are all valid. Then either of the edges $\overline{AB}$ and $\overline{BC}$ can be reduced to zero length by translating the infinite edge adjacent to it.*

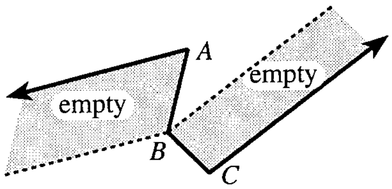


Figure 8: Each finite edge may be eliminated by translating an infinite edge

Proof: Let $\vec{e_A}$ be the infinite edge incident to A, and define $\vec{e_B}$ to be the ray parallel to $\vec{e_A}$ whose origin is B. Let $\vec{e_C}$ be the infinite edge incident to C. Edge $\overline{AB}$ can be shortened to zero length by translating $\vec{e_A}$ if and only if $\overline{BC}$ and $\vec{e_C}$ do not enter the infinite trapezoid bounded by $\vec{e_A}$, $\overline{AB}$, and $\vec{e_B}$. If edge $\overline{BC}$ enters the trapezoid, then angle $A+B$ is invalid; if the infinite edge $\vec{e_C}$ enters the trapezoid, then angle $A+B+C$ is invalid. By assumption, both angles are valid, so $\overline{AB}$ can be shortened. A similar argument for $\overline{BC}$ finishes the proof. □

The following lemma shows that a reduced form allows us enough flexibility to apply Lemma 4.

Lemma 5. *In a reduced form, the disk sizes can be chosen so that the infinite edges incident to any disk are free to translate a distance at least equal to the diameter of the disk.*

Proof: The proof is top-down. The base case is easily established: the top-level infinite edges of a chain are unconstrained. (In the top-level triangle of a polygonal reduced form, keeping the disk radius at most one-fourth of the minimum edge length of the triangle establishes the base case.) Within each disk $\mathcal{D}$ of a reduced form, there are two infinite edges f and g, joined by a finite edge e. The endpoints of e are contained in two subdisks concealing microstructure. In the inductive step, we assume that f and g are able to translate at least the diameter of $\mathcal{D}$, so they can certainly translate at least $\delta =$ the maximum diameter of a subdisk. Edge e appears infinite from the vantage point of each of its disks. When we translate e relative to the disk defined by (e, f), the other disk translates uniformly along with the intersection of the lines supporting e and g. Therefore, we choose δ small enough that translating e by δ relative to one subdisk does not move the other subdisk out of $\mathcal{D}$. □

To morph $((A\ B)\ C)$ to $(A\ (B\ C))$, we erase the disk around $\overline{AB}$, then translate the infinite edges to lengthen $\overline{AB}$ and shorten $\overline{BC}$, leaving vertex B (and its microstructure) fixed. Once $\overline{AB}$ is of finite length and $\overline{BC}$ is infinitesimally short, we draw a new disk around $\overline{BC}$, leaving A at the top level. During these translations and regroupings, we may scale the microstructure at A, B, and C uniformly if necessary. (Unless the infinite edges are parallel, we can actually do the rotation without translating the infinite edges at all. If they are parallel and we do have to move them, we make sure the movement is small enough that the condition of Lemma 5 is maintained at the other ends of those edges.)

Because a valid rotation on trees can be mapped directly to a constant-complexity morph of one reduced form to another, any algorithm for transforming trees via valid rotations corresponds directly to an algorithm for morphing the reduced forms of two parallel bi-infinite chains. Theorem 7 thus implies the following:

Theorem 6. *Given two parallel bi-infinite chains P and Q of n sides, we can continuously deform P to Q by parallel moves while maintaining simplicity. The deformation uses $O(n^{4/3+\epsilon})$ parallel moves.*

4 Tree reconciliation via rotations

In this section we address the purely combinatorial problem of transforming one binary tree into another via a sequence of (single) rotations. This is a well-studied problem [CW, STT]; it is known that $\Theta(n)$ rotations are necessary and sufficient to rotate one tree into another in the worst case (the best-known upper bound is $2n-6$). But in our setting there is additional structure to cope with. Our trees have *weights* at their leaves. The internal nodes also have induced weights, namely the sum of the leaf weights in all leaves belonging to the subtree rooted at that internal node. These weights restrict the possible trees we can have, as well as the allowed rotations between them.

As mentioned earlier, a weighted tree is called *valid* if the weights of all its nodes (leaves, as well as internal nodes) are reals in the range $(-1, +1)$ (open interval). A rotation is called valid if it transforms a valid tree into another valid tree. It is clear that rotations preserve the sequence of leaf weights, but change the grouping structure on these weights imposed by the tree. If T_1 and T_2 are two valid binary trees on n leaves with the same sequence of leaf weights $w_1, w_2, \ldots, w_n$, then the result mentioned in the previous paragraph asserts that one can always go from one tree to the other via a sequence of rotations. Arbitrary rotations, however, can destroy validity by generating internal nodes whose weights are outside the range $(-1, +1)$. So the natural question that arises is whether it is always possible to go from T_1 to T_2 via a sequence of valid rotations and, if so, what is the maximum required length of such a sequence.

We show that the answer to this question is affirmative, but we do not know if our upper bound is best possible (in fact, we strongly suspect that a nearly linear upper bound holds).

Theorem 7. *If T_1 and T_2 are any two valid binary trees on the same sequence of leaf weights $w_1, w_2, \ldots, w_n$, then it is always possible to go from T_1 to T_2 by a sequence of $O(n^{4/3+\epsilon})$ valid rotations.*

The rest of this section proves the theorem. Our basic strategy is go from T_1 to T_2 as follows: take any pair of leaves in T_2 that are siblings and perform valid rotations on T_1 (if necessary) to make them siblings in T_1 as well (we have to argue that this is always possible). In each tree contract that pair of leaves into their parent, making the parent a new leaf and giving it a weight that is the sum of the two previous leaf weights. We are now left with two valid trees on $n-1$ weighted leaves, and we repeat the whole process until both trees have been reduced to a single node. It is clear that in this process we generate a sequence of valid rotations that transforms T_1 into T_2. This simplistic scheme has to be refined by adding a certain "tree renormalization" step at key points, in order to speed up the process.

Let us then focus first on the problem of taking a pair of adjacent but non-sibling leaves w and y in T_1 and performing valid rotations to make them siblings. See Figure 9. Let x be the least common ancestor of w and y, and let v and z be the left and right children of x respectively. A key concept for our analysis is that of a *spine* in a binary tree. A spine is a maximal sequence of nodes that form a path in the tree all of whose links go in the same direction (all left or all right). For example, in Figure 9, the paths $[v..w]$ and $[z..y]$ are spines. We base our procedure for making w and y siblings on a sequence of *spine-contraction* steps, as in the lemma and corollary below.

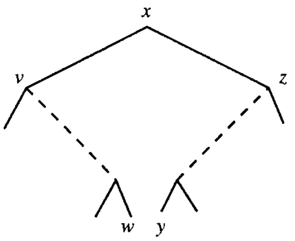


Figure 9: Two adjacent non-sibling leaves

Lemma 8. *Among any four consecutive internal nodes along a spine in a valid tree, at least one can be rotated out of the spine by a valid single rotation.*

Proof: Without loss of generality assume that the spine is a left spine, like $[z..y]$ in Figure 9. Let A, B, C, and D be the right children of four consecutive nodes on the spine, and let X be A's sibling, as in Figure 10a. By abuse of notation let A, B, C, D, and X also denote the weights of the nodes. If any consecutive pair of right children, say (A, B), has a valid sum $(-1 < A + B < 1)$, then a valid rotation can be applied at the edge joining the parents of A and B. This makes A and B into siblings whose grandparent lies on the spine, and removes one edge from the spine. See Figure 10b.

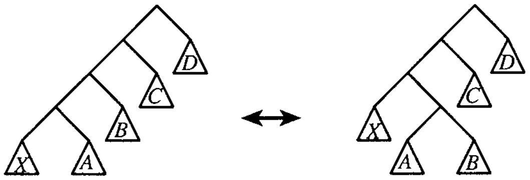


Figure 10: A rotation may be performed on any spine with more than four nodes

If no consecutive pair has a valid sum, then $|A+B| > 1$ and $|C+D| > 1$. All four of A, B, C, and D must have the same sign, so $|A+B+C+D| = |A+B|+|C+D| > 2$. The weight at the parent of D is $w = X+A+B+C+D$. Because $|X| < 1$, we have $|w| > 1$, so the initial tree was invalid, a contradiction. □

If we partition a spine into consecutive groups of four nodes each and apply the valid rotation guaranteed by the previous lemma to each group independently, we can reduce the length of our spine by $1/4$. By iterating this process until the previous lemma is no longer applicable, we obtain the following spine-contraction corollary. We note that when no rotation can be applied along a spine, all the weights of the spine nodes must be of the same sign.

Corollary 9. *Given a spine of length k, we can reduce it to length at most 4 by $k - 4$ valid rotations. Furthermore, when this procedure terminates, the original children of the spine will have been regrouped into trees attached to the contracted spine, each of height at most $O(\log k)$.*

If we use a spine contraction on the spines $[v..w]$ and $[z..y]$, we can make the least common ancestor x of leaves w and y be at most 5 levels above the two leaves. We can then finish as follows.

Lemma 10. *If the weights of two adjacent leaves w and y have a valid sum, then w and y can be made siblings by at most eight valid single rotations once the spines $[v..w]$ and $[z..y]$ have been contracted.*

Proof: After the spine-contraction process, we can assume that no rotation involving only spine $[v..w]$ or spine $[z..y]$ is valid. Let $l_1, \ldots, l_k$ denote the weights of the left children of spine $[v..w]$, in left-to-right order, and similarly let $r_1, \ldots, r_j$ denote the weights of the right children of $[z..y]$. Observe that the l's all have the same sign; the same is true for the r's. By abuse of notation, let w and y also denote the weights of the corresponding nodes. Denote the weight at x by $x = l_1 + \cdots + l_k + w + y + r_1 + \cdots + r_j$. We know $|x| < 1$.

If l_1 has the same sign as x, say positive, then $x > x - l_1 > x - 1 \geq -1$, so $x - l_1$ is valid. Similarly, if $\text{sign}(r_j) = \text{sign}(x)$, then $x - r_j$ is valid. If $\text{sign}(l_1) = \text{sign}(r_j) \neq \text{sign}(x)$, then $\text{sign}(x) = \text{sign}(w + y)$, say positive. Then $x < x - l_1 < w + y < 1$ and $x - l_1$ is valid (so is $x - r_j$).

Without loss of generality assume $x - l_1$ is valid. Perform a rotation at edge (v, x), moving the right child of v to be the left sibling of z. The new parent of z and the moved node has weight $x - l_1$, which is valid. All the other modified nodes have weights that were present before the rotation, so the rotation is valid. It shortens by one the length of the path between w and y. After the rotation, we can still apply the argument above to show that at least one of the edges incident to the lowest common ancestor of w and y can be rotated. Performing at most eight rotations, we shorten the path length to two, making w and y siblings. □

The overall rotation cost of making leaves w and y siblings is proportional to the sum of the original lengths of the two spines $[v..w]$ and $[z..y]$. This is 2 less than the tree distance of w and y. Let us call the latter quantity simply the *distance* in T_1 between w and y.

Our discussion so far, coupled with our overall tree morphing strategy mentioned earlier, provides a proof that T_1 can be rotated into T_2. In fact, we also get an $O(n^2)$ upper bound on the number of rotations needed, as the distance between any two adjacent leaves is $O(n)$, and the number of sibling-formation operations is at most $n - 1$.

In order to improve this bound, we must analyze more carefully the distances between adjacent leaves that are being made siblings. In particular, the difficulty is that rotations that reduce one spine length can increase other spine lengths—actually it is very easy to check that any single rotation will decrease the distance between two pairs of leaves by 1, and it will increase the distance between two other pairs by 1. We can, however, argue that there is a sequence of spine contractions over a whole tree that reduces the maximum spine length.

Lemma 11. *Any valid tree on n leaves with maximum spine length k can be rotated using $O(n)$ rotations into another valid tree whose maximum spine length is $O(\log k)$.*

Proof: Apply Corollary 9 on the left and right spines of the original tree, reducing them to constant length. Each subtree that was originally a child of the left or right spine is now at depth $O(\log k)$. Let T be such a subtree. Without loss of generality, assume that the parent of T is to its left. Apply Corollary 9 on the right spine of T, reducing it to length at most four. (This process does not modify any ancestors of T.) Now the length of the right spine that includes T's right spine (including ancestors) is $O(\log k)$. Recursively apply this procedure to the original child trees of T's right spine, until every node in the tree has participated in one spine-reduction operation.

Spine-reduction does not affect the internal structure of the children of the spine; the cost of spine-reduction is proportional to the length of the spine. Each spine-reduction operation is applied to a spine that has not previously been operated upon. Thus the total work is $O(n)$. By construction, each spine in the final tree consists of at most four nodes that result from the reduction of that spine, plus $O(\log k)$ ancestors that were rotation nodes in a spine reduction higher up in the tree. □

Corollary 12. *Any valid tree on n leaves can be rotated using $O(n \log^* n)$ rotations into another valid tree whose maximum spine length is $O(1)$.*

Proof: Recall that $\log^* n = \min(k \mid \log^{(k)} n \leq 1)$, where $\log^{(k)} n$ is the k-fold composition $\log(\log \cdots (n))$. By applying Lemma 11 $O(\log^* n)$ times, we reduce the maximum spine length to $O(1)$. □

We call the process of transforming a tree into one whose maximum spine length is constant a *normalization.* We intersperse normalizations with the leaf pairing and contraction process so as to keep leaf distances small. But we need to develop one more idea first, the intuition being that during the pairing and contraction process some leaf pairs may have a large distance, but not many may do so. After all, in a single tree, the average distance between adjacent leaves is constant—but of course our rotations change the tree all the time.

We saw earlier that rotations redistribute spine length. In fact, Corollary 9 states that the rotations that make siblings of a pair of leaves that were originally at distance k can increase the distance of any other adjacent pair of leaves by at most $O(\log k)$. Let us consider the following game: we have n bins, each containing at most n tokens. The bins model the gaps between successive leaves, and the tokens account for the distances between successive pairs. We play as follows:

- At each turn, we remove all k tokens from one bin and distribute them to k other bins. At most one new token gets added to each bin at each turn.
- We play for n turns; the goal is to maximize the number of tokens moved.

In the real tree situation, contracting a pair of leaves at distance k may increase the distance of some other pair not by 1 but by $O(\log k)$. But let us focus on the simpler game situation for now.

Lemma 13. *If originally each bin contains $O(1)$ tokens, then after $n^{2/3}$ turns, the maximum number of tokens moved is $\Theta(n)$.*

Proof: We first show that during any run of $n^{1/3}$ turns in the first $n^{2/3}$ turns, at most $n^{2/3}$ tokens are moved. The run moves tokens from a set B of $n^{1/3}$ bins into other bins. There are at most $n^{2/3}$ tokens in B at the start of the run (because we are in the first $n^{2/3}$ turns) and at most $n^{2/3}$ tokens get added into B during the run (because each bin in B receives at most $n^{1/3}$ tokens in $n^{1/3}$ turns). Partitioning the first $n^{2/3}$ turns into $n^{1/3}$ runs of $n^{1/3}$ turns apiece completes the upper bound.

To prove the lower bound, let A and B be two disjoint sets of $n^{1/3}$ bins. In the first $\frac{1}{2}n^{2/3}$ turns, put at least $\frac{1}{2}n^{1/3}$ tokens in each bin of A. Divide the second $\frac{1}{2}n^{2/3}$ turns into $\frac{1}{2}n^{1/3}$ phases of $n^{1/3}$ turns apiece. In each phase, move all the tokens from A to B or vice versa, putting $\frac{1}{2}n^{1/3}$ in each bin. □

In the slightly more elaborate tree setting in which a single contraction of length k may increase a particular distance by $O(\log k)$, the bound on the total work done after $n^{2/3}$ contractions becomes $O(n^{1+\epsilon})$.

We can now complete the proof of the main theorem of this section. After every $n^{2/3}$ contraction steps, we renormalize the tree. Thus the cost of the rotations for the leaf contractions balances the cost of the rotations for renormalization and both are bounded by $n^{4/3+\epsilon}$. Thus the theorem is proved.

5 Morphing polygons

In this section we use the tools of the preceding sections to show how to morph one polygon to another parallel polygon. This is more difficult than morphing bi-infinite chains (Sections 3 and 4), because we cannot guarantee that the reduced forms of two parallel polygons will have the same top-level triangle. We show below that each morphing operation that involves two of the top-level trees can be transformed with $O(1)$ parallel moves to involve a single tree.

Let $\mathcal{T}_1 = (T_1, T_1', T_1'')$ be the set of three trees associated with the top-level triangle of the reduced form of polygon P, and let $\mathcal{T}_2 = (T_2, T_2', T_2'')$ be the corresponding set of trees for Q. In the tree model, our algorithm for morphing P to Q is the same as that of Section 4: we pick a pair of leaves that are siblings in one of the trees of $\mathcal{T}_2$, rotate them to be siblings in one of the trees of $\mathcal{T}_1$, and then collapse the pair into their parent in both $\mathcal{T}_1$ and $\mathcal{T}_2$, thereby reducing the size of the problem. This works fine if the two leaves belong to the same tree in $\mathcal{T}_1$: we simply apply the algorithm of Section 4 to that tree. If the two leaves belong to different trees, however, we must morph the polygon to bring the two leaves into the same tree before we can apply the algorithm of Section 4.

Let x and y be two sibling leaves in $\mathcal{T}_2$, and without loss of generality suppose that x and y are the last and first leaves in T_1 and T_1', respectively. We begin the morph by shortening the spines containing x and y as much as possible (Corollary 9). Then the right spine of T_1 has at most four left children whose weights $l_1, \ldots, l_k$ all have the same sign. The left spine of T_1' has at most four right children with same-sign weights $r_1, \ldots, r_j$. The weight of T_1 is $l_1 + \cdots + l_k + x$, and the weight of T_1' is $y + r_1 + \cdots + r_j$. Because T_1 and T_1' are both top-level trees, their weights are both positive, and their sum is between 1 and 2. Leaves x and y are siblings in $\mathcal{T}_2$, so their combined weight is valid: $|x + y| < 1$. The following lemma is our basic tool for separating a top-level tree into two:

Lemma 14. *Let T be a valid tree, and suppose the left (right) spine of T has $O(1)$ children, all of whose weights are positive. Let e be a spine edge such that the total leaf weight to the left of e is valid, as is the total weight to the right. Then the polygonal chain corresponding to T can be morphed with $O(1)$ parallel moves to separate it into two linearly separated subchains joined by a single edge, corresponding to splitting T at e.*

Proof: The children of the spine above e are all positive. Let T_e be the subtree of T rooted at the bottom of e. T_e corresponds to a microstructure disk $\mathcal{D}_e$ in the reduced form. If we erase all the microstructure disks that contain $\mathcal{D}_e$, we obtain a mostly-convex chain with disks at its vertices (the only possible nonconvexity is at $\mathcal{D}_e$). See Figure 11. We coalesce all the disks except $\mathcal{D}_e$ together by parallel moves (because T is valid, none of the requisite edge-shortenings causes a collision between chain edges.) This essentially leaves a two-vertex valid chain, whose finite edge can be stretched arbitrarily. □

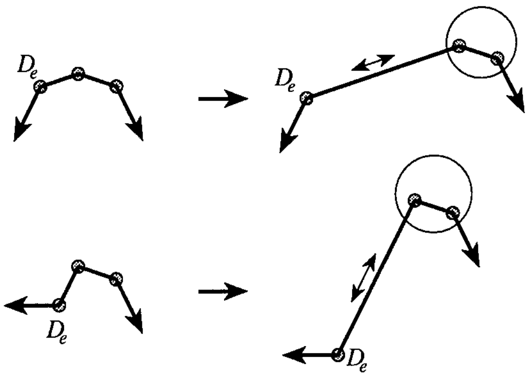


Figure 11: Splitting a chain into two separated chains

Consider the case in which the sequence of weights $l_1, \ldots, l_k, x, y, r_1, \ldots, r_j$ can be split into two subsequences, each with a valid sum, without separating x and y. Without loss of generality assume the split is to the right of some l_i. The sum $l_1 + \cdots + l_i$ is positive, and the sum $W = l_{i+1} + \cdots + x$ must be valid. Thus by Lemma 14 the chain corresponding to T_1 can be split in two by stretching the edge e corresponding to the split point of the sequence. We perform a parallel move, stretching e and bringing the subchain corresponding to W closer to the chain (that is, microstructure) for T_1'. Let f be the edge of the top-level triangle joining T_1 to T_1'. If W is positive, we translate f away from the triangle interior, so that it gets shorter (Figure 12a). If W is negative, we hold f stationary and translate e outward, again shortening f (Figure 12b). In both cases the parallel move does not encroach upon the triangle's third vertex. The move shortens f enough that we can draw a microstructure disk around it: the triangle has been changed so x and y belong to the same tree, and the algorithm of Sections 3 and 4 applies.

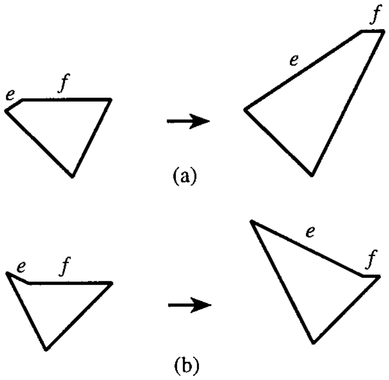


Figure 12: Modifying the top-level triangle in one step

Lemma 15. *The sequence $l_1, \ldots, l_k, x, y, r_1, \ldots, r_j$ can be split into at most three subsequences, each with a* *valid, positive sum, with x and y in the same subsequence.*

Proof: Consider the modified sequence $l_1, \ldots, l_k, (x + y), r_1, \ldots, r_j$. Without loss of generality assume the l's are positive. Let $\mathcal{P}$ be the longest prefix of the modified sequence whose sum is valid. The sum of $\mathcal{P}$ is positive (the sum up to y is > -1, and if that sum is negative, the b's must be positive).

If the suffix after $\mathcal{P}$ is valid, and hence positive, we are done. If not, we split the suffix after its first element. This element is positive and valid. The sum of the last part of the suffix is also positive (else the whole suffix would be valid) and valid (else the total sequence sum would be > 2). □

If the weight sequence cannot be split into two valid pieces without separating x and y, we split it in three pieces, as in Lemma 15. We stretch the edge corresponding to one of the two splits, turning the triangle into a convex quadrilateral with edge $f = \overline{xy}$ separated from the other two top-level vertices. See Figure 13. (The quadrilateral must be convex, else the weight sequence could be split into two valid pieces.) By translating the quadrilateral edge opposite f outward, we shrink it enough to draw a microstructure disk around it. (Translating outward shrinks the edge because the top-level turn at the endpoints of f sums to more than > 1.) Now we stretch the edge corresponding to the other split. As in Figure 12a, we translate f outward, shortening it until it can be contained in a microstructure disk. Now x and y belong to the same tree, and the previous algorithm applies.

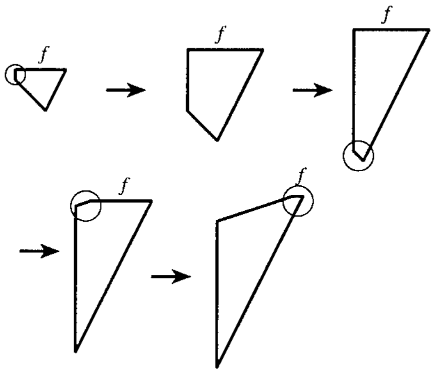


Figure 13: Modifying the top-level triangle in two steps

Except for the preliminary spine-shortening, we use only $O(1)$ parallel moves to make x and y into siblings in the same tree. The game model of Lemma 13 still applies to morphing polygons: bins still correspond to gaps between leaves, but now adjacent leaves may be in different trees. Thus the number of parallel moves needed to morph polygons is asymptotically the same as the number needed for bi-infinite chains:

Theorem 16. *Given two parallel simple polygons P and Q of n sides, we can continuously deform P to Q by parallel moves while maintaining simplicity. The deformation uses $O(n^{4/3+\epsilon})$ parallel moves.*

6 Open questions

The problem discussed in this paper and our solution to it raise many fascinating questions.

The two specific questions left open by our approach are (1) to show that in any polygon an edge can always be eliminated by an edge translation, and (2) to get a tight bound on the number of parallel moves needed for the morph. As mentioned earlier, we strongly suspect that this number is nearly linear.

More generally, it would be interesting to introduce some measure of polygon distortion and look at edge translation algorithms that minimize this measure. Our algorithm is very bad this way, for even if P and Q are very similar-looking polygons, their intermediate morphs can visually look quite different.

Acknowledgements: The authors wish to acknowledge valuable discussions with John Hughes, Tom Shermer, David Kirkpatrick, G.D. Ramkumar, and Emo Welzl.

References

[CR] J. Culberson and G. Rawlins. Turtlegons: generating simple polygons from sequences of angles. In *Proceedings of 1985 Comp. Geom. Symp.*, pages 305–310, 1985.

[CW] K. Culik and D. Wood. A note on some tree similarity measures. In *IPL*, 15, pages 39–42, 1982.

[KR] A. Kaul and J. Rossignac. Solid-interpolating deformations. In *Proceedings of Eurographics 1991*, pages 494-505, 1991.

[SGWM] T. Sederberg, P. Gao, G. Wang, and H. Mu. 2-d shape blending: an intrinsic solution to the vertex path problem. In *Computer Graphics*, 27, pages 15–18, 1993 (SIGGRAPH '93)

[SG] T. Sederberg and E. Greenwood. A physically based approach to 2-d shape blending. In *Computer Graphics*, 26, pages 25–34, 1992 (SIGGRAPH '92)

[STT] D. Sleator, R. Tarjan, and W. Thurston. Rotation distance, triangulations, and hyperbolic geometry. In *Proceedings of the 1986 STOC Symposium*, pages 122-135, 1986.

[T] C. Thomassen. Deformations of planar graphs. In *J. Comb. Theory Ser. B*, 34, pages 244–257, 1983.

[V] G. Vijayan. Geometry of planar graphs with angles. In *Proceedings of 1986 Comp. Geom. Symp.*, pages 116–124, 1986.

[W] E. Welzl. Personal communication, 1993.

A New Technique to Compute Polygonal Schema for 2-manifolds with Application to Null-homotopy Detection

Tamal K. Dey
Dept. of Computer Science
Indiana University-Purdue University at Indianapolis

Abstract

We provide a new technique for deriving optimal sized polygonal schema for triangulated compact 2-manifolds without boundary in $O(n)$ time, where n is the size of the given triangulation T. We first derive a polygonal schema P embedded in T using Seifert-Van Kampen's theorem. A reduced polygonal schema Q of optimal size is computed from P, where a surjective mapping from the vertices of P is retained to the vertices of Q. This helps detecting null-homotopic (contractable to a point) cycles. Given a cycle of length k we determine if it is null-homotopic in $O(n+gk)$ time where g is the genus of the given 2-manifold. The actual contraction for a null-homotopic cycle can be computed in $O(gkn)$ time and space. This is an improvement of a factor of g over the previous best-known algorithms for these problems.

1 Introduction

In recent years a new focus has developed in studying the algorithmic aspects of topology [2, 4, 6, 7, 9, 10], a well-developed branch of mathematics. This emergent field has been called "Computational Topology" [6, 10]. It is generally recognized that there exists a vast repository of topological problems which have not been studied extensively from an algorithmic point of view. This paper deals with the problem of computing polygonal schema, an efficient representation of 2-manifolds, from their triangulations. An elegant solution to this problem provides an improved algorithm for detecting null-homotopic cycles on compact 2-manifolds.

A 2-manifold $\mathcal{M}$ can be represented by a polygon P (polygonal schema) with even number of edges such that when edges are appropriately identified, we get back $\mathcal{M}$. Given a triangulation T of $\mathcal{M}$, Vegter and Yap [10] designed an algorithm that derives a polygonal schema (*canonical*) of optimal size whose edges belong to the edges of a refined triangulation of T. This method subdivides the edges of T into $O(g)$ subedges and increases the size of the triangulation to $O(gn)$. A lower bound of $\Omega(gn)$ is known for deriving such a canonical polygonal schema for 2-manifolds. In that respect the algorithm of Vegter and Yap is optimal. However, in applications we may not need such polygonal schema. As shown in this paper, we can derive a polygonal schema of optimal size in $O(n)$ time from T without refining it. This polygonal schema is not embedded in a refinement of T but retain a surjective mapping from the vertices of another polygonal schema embedded in T to its vertices. This fact is used to detect null-homotopic cycles on $\mathcal{M}$. Our method of computing the polygonal schema is also simpler than Vegter and Yap's method. We believe that this method will find more applications in other algorithms.

Given a cycle C on a 2-manifold, we say that C is null-homotopic if it can be contracted to a single point. Detecting the null-homotopy of a given cycle in a topological space is a century-old problem, known as the *contractability problem* in topology [8]. A related problem involves determining if two given cycles can be continuously deformed to one another. It turns out that a solution to the contractability problem also provides a solution to this problem.

The contractability problem for 2-manifolds was first solved from a mathematical point of view during the 1880s. It was established that a cycle is null-homotopic if and only if its corresponding curve on the *universal covering space* is closed. Universal covering space is a collection of polygons (polygonal schema) appropriately attached to give a tessellation of the plane. In [6], Schipper used this result to give an $O(g^2k+gn)$ time and space algorithm to detect the null-homotopy

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

of a given cycle. Here k is the length of the given cycle. In his algorithm Schipper derives the canonical polygonal schema using the algorithm of Vegter and Yap [10]. Using our polygonal schema computation we design an algorithm that runs in $O(n+gk)$ time and space. If the given cycle is null-homotopic, the actual contraction can be computed in $O(gnk)$ time. This is an improvement of a factor of g over Schipper's algorithm. We should mention that Schipper's algorithm does not remain linear in k and takes $O(n+k^2)$ time when $g=1$ for torus (orientable) and $g=2$ for klein bottle (non-orientable). We treat these cases separately and give $\Theta(n+k)$ optimal algorithms for them.

We describe necessary concepts in homotopy and fundamental groups of topological spaces in Section 2. In the third section we explain the polygonal representation (polygonal schema) of 2-manifolds. In Section 4 we describe a technique to derive a reduced polygonal schema from the given triangulation of a 2-manifold using Seifert-Van Kampen's theorem. In the fifth section we detail the algorithm for null-homotopy detection, and we conclude in Section 6.

2 Homotopy and Fundamental Group

Let $f : T_1 \rightarrow T_2$ and $g : T_1 \rightarrow T_2$ be two maps between the topological spaces T_1 and T_2. These two maps are called homotopic if there is a continuous function $h : [0,1] \times T_1 \rightarrow T_2$ such that $h(0,x) = f(x)$ and $h(1,x) = g(x)$. We can interpret h as a deformation process that transforms f to g in a continuous manner. We are interested in the case when $T_1 = S^1$, the unit circle. Then, f and g are closed curves, also called cycles. Let C_1, C_2 be two cycles that begin at a common point, say p. The product operation, " $\cdot$ ", is defined on them as $C_3 = C_1 \cdot C_2$, where C_3 is the cycle beginning at p, then going around C_1 followed by a traversal around C_2 which finishes at p. The inverse C^{-1} of a cycle C is the cycle lying on top of C, but with opposite orientation.

Given a fixed origin p for the cycles on T_2, we call C, C' equivalent if there is a homotopy between them which keeps p fixed. The equivalence class of C is denoted by $[C]$. The product operation " $\cdot$ " extends naturally to equivalence classes as $[C_1] \cdot [C_2] = [C_1 \cdot C_2]$.

These equivalence classes form a *group* under the product operation. The identity element 1 is represented by the equivalence class of the point cycle p, and $[C] \cdot [C^{-1}] = [C \cdot C^{-1}] = 1$ giving $[C]^{-1} = [C^{-1}]$. This group, denoted $\pi_1(T_2)$, is called the *fundamental group* of the topological space T_2. It turns out that this group is independent of the choice of the origin p and is an invariant property of the underlying space.

Fact 2.1 *A cycle C is contractable or null-homotopic if and only if $[C] = 1$ in π_1 [8].*

We use the following concepts of group theory in the next sections. Let G be a group with the product operation "$\cdot$". A set of elements $X = \{g_1, g_2, ...\}$ of G is called a *generator set* if any non-identity $g \in G$ can be written as $g = g'_1 \cdot g'_2 ... g'_k$ for some $k \geq 1$ and $g'_i \in \{g_1, g_2, ...\}$. In other words all elements of G are *generated* by X. A *word* is a concatenation of elements of G under the product operation. For example $w = g_1 . g_2 . g_3 = g_1 g_2 g_3$ is a word. A *relation* r is a word which is set to 1. For example $r = g_1 g_2 g_3$ is a relation if $g_1 g_2 g_3 = 1$. Relations of the form $gg^{-1}, g^{-1}g$ are called *trivial* relations.

The structure $\mathcal{F} =< g_1, g_2 ... : r_1, r_2, ... >$ is called a *group presentation* of G if $\{g_1, g_2, ...\}$ is a generator set that generates G with respect to the relations $\{r_1, r_2, ...\}$. We also write $G =< g_1, g_2, ... : r_1, r_2 ... >$. Two relations are *equivalent* if one can be derived from the other. A group is *freely generated* by $X = \{g_1, g_2, ...\}$ if there are no nontrivial relations. In that case the group has a presentation of the form $< g_1, g_2, ... : -- >$.

3 Polygonal Schema

A *2-manifold* is a topological space where each point has a neighborhood homeomorphic to an open disk. By this definition, we are considering only 2-manifolds without boundary. A 2-manifold can be infinite or finite. Moreover, it can be closed or open depending on whether or not its closure coincides with itself. A closed and finite 2-manifold is also called a compact 2-manifold. A sphere and a klein bottle are two examples of compact 2-manifolds. A 2-manifold is called orientable if it has two distinct sides. Otherwise, it is non-orientable. For details see [8].

A 2-manifold is triangulable in the sense that it can be represented as the union of a set of triangles, edges, and vertices satisfying the following properties. Each pair of triangles either share a single vertex or a single edge, or are completely disjoint. Also, the triangles incident on a vertex can be ordered circularly so that two triangles share a common edge if and only if they are adjacent in this ordering. In this paper we consider only compact 2-manifolds without boundary.

Any orientable or non-orientable 2-manifold $\mathcal{M}$ can be represented by a simple polygon P with an even number of edges on $bd(P)$[1] which is also called a *polygonal schema* of $\mathcal{M}$. Each edge of P has a signed label such that each unsigned label occurs twice. See [3, 8]

[1] $bd(P)$ represents the boundary of the polygon P.

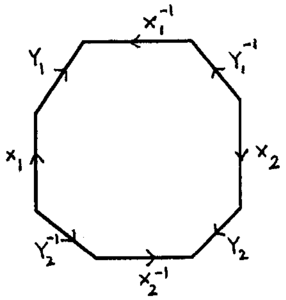


Figure 3.1: A polygonal schema for double torus

for details. Two edges with the same unsigned labels are called *partnered edges.* Partnered edges can have labels with the same or opposite signs. Two partnered edges with the labels $+x$ and $-x$ represent the same edge on $\mathcal{M}$ but are oppositely directed on P. Figure 3.1 shows a polygonal schema for double torus ($g = 2$). We use x^{-1} to denote the complement of the label x. To reconstruct a surface homeomorphic to $\mathcal{M}$ from its polygonal representation, the oriented edges with the same labels are identified together in such a way that their orientations match. For simplicity, we say that $\mathcal{M}$ is obtained from P by identifying partnered edges appropriately.

An orientable 2-manifold $\mathcal{M}$ with genus $g > 0$ can be represented *canonically* using a $4g$-gon where all $4g$ vertices represent the same vertex on $\mathcal{M}$. The size of this polygon is *optimal* since no k-gon with $k < 4g$ can represent $\mathcal{M}$. The labels on the edges around the polygon are of the form:

$$x_1y_1x_1^{-1}y_1^{-1}x_2y_2x_2^{-1}y_2^{-1}...x_gy_gx_g^{-1}y_g^{-1}.$$

Similarly, a non-orientable 2-manifold with genus $g > 1$ can be represented *canonically* using an optimal sized $2g$-gon where the labels on the edges around the polygon are of the form:

$$x_1x_1x_2x_2...x_gx_g.$$

For $g = 0$, the orientable 2-manifold is the sphere which can be represented canonically by two directed edges xx^{-1}. Similarly for $g = 1$, the non-orientable 2-manifold is the projective plane which can be represented by two directed edges xx. If we identify the partnered edges of the canonical polygon appropriately, they form a set of curves glued at a single point on the 2-manifold. These curves are called *canonical generators.*

Let T be a triangulation of a 2-manifold $\mathcal{M}$ with n edges. A path on T is a sequence of alternating vertices and edges, $v_1e_1v_2e_2...e_{k-1}v_k$, where the edge e_i has vertices v_i, v_{i+1} as endpoints. A cycle is a path, which is closed.

The following lemma provides a method that flattens out T to a triangulation T' of a planar polygon.

Lemma 3.1 *A polygonal schema P with triangulation T' can be constructed from T where there is a one-to-one correspondence between triangles of T' and T.*

PROOF. We construct a sequence of closed disks $D_1, D_2, ..., D_n$ in a plane incrementally such that $P = D_n$ at the end. Initially, $D_1 = \sigma'_1$, a triangle in the plane that corresponds to an arbitrarily chosen triangle σ_1 on T. Let $D_i = \sigma'_1 \cup \sigma'_2 ... \cup \sigma'_i$ after the ith step. At the $i+1$ th step we choose a triangle σ_{i+1} on T which has the following properties: (i) no triangle corresponding to σ_{i+1} has been included in D_i, and (ii) a triangle σ_j adjacent to σ_{i+1} by an edge has a corresponding triangle in D_i. These two conditions imply that there is an edge $e = \sigma_j \cap \sigma_{i+1}$ such that its corresponding edge e' on D_i appears on $bd(D_i)$. We attach a triangle σ'_{i+1} to $bd(D_i)$ such that $\sigma'_{i+1} \cap bd(D_i) = e'$. This gives the new disk $D_{i+1} = D_i \cup \sigma'_{i+1}$. It is clear that if D_i is a closed disk, so is D_{i+1}. Finally, when we exhaust all triangles on $\mathcal{M}$, we have D_n with the triangulation T' that has the following properties. (i) Each triangle in T' corresponds to a single triangle in T and vice versa. (ii) Each edge e' on $bd(D_n)$ has a partnered edge e'' such that they both correspond to a single edge e on T. This is because each edge e on T has two incident triangles and the edges on $bd(D_n)$ have a single triangle incident on them. (iii) When partnered edges of $bd(D_n)$ are identified, we get back T. This is because, by our construction, the two triangles in T' incident on partnered edges e', e'' correspond to the two triangles incident on e in T. Incidence relations of all other edges inside T' are isomorphic to their corresponding edges on T. So, $D_n = P$, a polygonal schema for $\mathcal{M}$ with a one-to-one correspondence between triangles in T and T'.

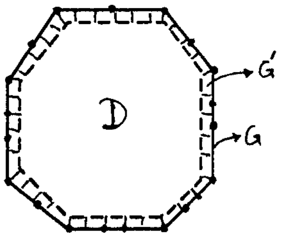


Figure 4.1: The spaces D, G and G'

4 Computing an optimal polygonal schema

The size (number of edges on the boundary) of the polygonal schema as deduced in the last section can be $\Omega(n)$, which is too large for our purpose. Here, we show a technique to derive a polygonal schema Q of optimal size from P that retains enough information to preserve a surjective mapping from vertices of P to Q. Due to this mapping, we would be able to trace a path on Q that corresponds to a path on P and hence a path on T. We already observed that the triangulation T' of P becomes isomorphic to the triangulation T of $\mathcal{M}$ when partnered edges are identified. From now on we will not distinguish between the triangles, edges of T and T'; we refer to them with the same name. With this set up, two partnered edges on $bd(P)$ have the same name.

Let the 1-complex (graph) formed by the identified edges of $bd(P)$ be G. The space $D = \mathcal{M} - G$ is an open disk since $P - bd(P)$ is an open disk. Let us attach a thin layer of an open set on two sides of the edges of G to make it open, and denote this space G'; see figure 4.1. We can express $\pi_1(\mathcal{M} = G' \cup D)$ in terms of $\pi_1(G')$ and $\pi_1(D)$ using the following theorem:

Seifert-Van Kampen's theorem: Suppose S is a space which can be expressed as the union of path-connected open sets A, B such that $A \cap B$ is path connected and such that $\pi_1(A)$ and $\pi_1(B)$ have respective representations

$$\pi_1(A) =< a_1, a_2, ... : r_1, ..., r_n >$$
$$\pi_1(B) =< b_1, b_2, ... : s_1, ..., s_q >$$

while $\pi_1(A \cap B)$ is finitely generated.

Then $\pi_1(S) =< a_1, a_2, ..., b_1, b_2.... : r_1, ..., r_n, s_1, ..., s_q, u_1 v_1^{-1}, ..., u_m v_m^{-1} >$ where u_i, v_i are the expressions for the same generator of $\pi_1(A \cap B)$ in terms of the generators of $\pi_1(A)$ and $\pi_1(B)$, respectively.

Since any cycle (curve) on G' can be continuously deformed to the edges of G, we have $\pi_1(G) = \pi_1(G')$. Let a set of generators of $\pi_1(G)$ be γ with the set of relations ρ_1. The fundamental group of D is the trivial group $\{1\}$ since D is an open disk. Let the set of relations obtained for the generators of $\pi_1(D \cap G')$ in terms of the generators of $\pi_1(D)$ and $\pi_1(G')$ be ρ_2. The spaces $\mathcal{M}$, G' and D satisfy all conditions of Seifert-Van Kampen's theorem. Then according to this theorem $\pi_1(\mathcal{M})$ has a presentation $< \gamma : \rho_1, \rho_2 >$.

A presentation of $\pi_1(G)$ can be obtained as follows. Let Y be a spanning tree of G, and p be a vertex of Y. Each edge e of G defines a cycle $c(e) = wew'$ originating at p where w, w' are the paths from p to the end points of e along the edges of Y. To simplify notations we write e for the cycle $c(e)$. Any cycle e where e is an edge of Y is contractable to p and hence belongs to the identity of $\pi_1(G)$. Let $B = \{b_1, b_2, ..., b_\ell\}$ be the set of edges of G that do not belong to Y. A presentation of $\pi_1(G)$ is $< b_1, b_2, ..., b_\ell : -- >$ implying $\rho_1 = \{\}$. Consider the sequence of edges around (clockwise) $bd(P)$. Some of these edges belong to Y which are set to 1 in $\pi_1(G)$. In terms of the generators of $\pi_1(G)$, let the sequence of edges around $bd(P)$ form the word $b_1'11...b_2'11...b_3'11...b_{2\ell}'11...1$, where b_i's represent the edges in B or their inverses. This word is equivalent to $r = b_1'b_2'...b_{2\ell}'$. The generator for the space $D \cap G'$ is $b_1'b_2'...b_{2\ell}'$ in terms of the generators of $\pi_1(G')$ and is 1 in terms of the generator of D. So we have $\rho_2 = b_1'b_2'...b_{2\ell}' = 1$. Hence a presentation of $\pi_1(\mathcal{M})$ is $< b_1, b_2, ..., b_\ell : \rho_2 >$.

Lemma 4.1 *$\ell = 2g$ if $\mathcal{M}$ is orientable and $\ell = g$ if $\mathcal{M}$ is non-orientable.*

PROOF. Consider the polygon P with all edges in Y contracted to a single point. This results in a polygon Q with the edges $b_1'b_2'...b_{2\ell}'$ labeled clockwise around it (figure 4.2). The polygon Q is a 2ℓ-gon with partnered edges. Further we can assume that all vertices of Q have the same label, i.e., when partnered edges are identified, all vertices are identified to a single point. Then Q represents a polygonal schema for a manifold $\mathcal{M}'$ homeomorphic to $\mathcal{M}$. This is because (i) When partnered edges are identified Q form a 2-manifold $\mathcal{M}'$ whose fundamental group is same as $\pi_1(\mathcal{M})$ (apply Seifert-Van Kampen's theorem) and (ii) any two compact 2-manifolds are homeomorphic if and only if their fundamental groups are isomorphic. Euler's characteristic of $\mathcal{M}'$ is $2 - \ell$ since we get one vertex, ℓ edges and one face when the edges of Q are identified. It is known that the Euler's characteristic of a surface $\mathcal{M}$ of genus g is $2 - 2g$ if $\mathcal{M}$ is orientable and is $2 - g$ if $\mathcal{M}$ is non-orientable. Since $\mathcal{M}$ and $\mathcal{M}'$ are homeomorphic, their Euler's characteristics must be equal, proving $\ell = 2g$ if $\mathcal{M}$ is orientable and $\ell = g$ if $\mathcal{M}$ is non-orientable.

4.1 The Algorithm

We construct the polygonal schema P as described in lemma 3.1. If we assume that the triangulation T is represented with a data structure that allows us to access the triangles incident on an edge and the edges of a triangle in constant time, then the construction of P takes $O(1)$ time per triangle and $O(n)$ in total. In case T is represented with only a set of triangles such that the edges of a triangle are accessed in constant time but not the triangles incident on an edge, we spend $O(n)$ extra time to preprocess this data structure. If the edges are given with integer names, we index the set of edges of the given triangles in an array and match them to detect the incident triangles for each edge. If the edges are given as pair of vertices, we sort them through radix sort in linear time and then match them.

While constructing P we maintain pointers from the edges and triangles of T to the edges and triangles of P. Also, the edges of the 1-complex G are detected during this step. A spanning tree Y of G is computed in $O(n)$ time. The edges not in Y are detected and the corresponding edges on $bd(P)$ are marked with the label other than 1. All other edges on $bd(P)$ are marked 1. All these take $O(n)$ time.

The reduced polygon Q of size 2ℓ is constructed from P as follows. Let the sequence of edges around $bd(P)$ (clockwise) be $b'_1 b'_2 ... b'_{2\ell}$ ignoring the edges marked 1. We form the polygon Q to have 2ℓ edges labeled $b'_1 b'_2 ... b'_{2\ell}$ around it. We maintain pointers from the vertices of P to the vertices of Q as follows. Let $b'_i b'_{i+1}$ be any two consecutive edges in the sequence (circular) $b'_1 b'_2 ... b'_{2\ell}$ and $v_1 v_2 ... v_s$ be the vertices of $bd(P)$ between b'_i and b'_{i+1} where v_1 is an endpoint of b'_i and v_s is an endpoint of b'_{i+1}. All these vertices point to the same vertex v between b'_i and b'_{i+1} in Q. See figure 4.2. The polygon Q can be thought of as the polygon P with all edges in the spanning tree Y shrunk to a single vertex. By lemma 4.1 Q has optimal size. The pointers from P to Q realizes a surjective mapping between their vertices. Combining all costs together Q can be constructed from T in $O(n)$ time.

Theorem 4.2 *Let $\mathcal{M}$ be any compact 2-manifold of genus g with triangulation T of size n. A polygonal schema Q of optimal size can be constructed in $O(n)$ time, where a surjective mapping is retained from the vertices of a polygonal schema embedded in T to the vertices of Q.*

5 Detecting Null-Homotopy

Traversing a cycle C on T is equivalent to traversing an ordered set of paths $U = \{u_1, u_2, ..., u_r\}$ that have end points on $bd(P)$ if C intersects G. Each of these paths u_i is homotopic to a path u'_i that has edges only on $bd(P)$ and have the same endpoints as u_i. Note that no such path exists when C does not intersect G. In that case C is trivially null-homotopic. We consider the nontrivial case when C intersects the edges of G.

When edges on $bd(P)$ are identified, the sequence $u'_1 u'_2, ..., u'_r$ form a cycle C' on T, which is homotopic to C. Each path u'_i can be expressed as a word w_i in terms of the generators $b_1, b_2, ..., b_\ell$ by listing the sequence of edges on it that are not marked 1 on $bd(P)$. This is actually done on $bd(Q)$ to avoid unnecessary visits of edges that are marked 1 on $bd(P)$. We also write $w_i \sim u_i$ if the word w_i corresponds to the path u_i. The word $w = w_1 w_2 ... w_r$ represents the cycle C in $\pi_1(\mathcal{M})$. We determine if $w = 1$ to detect if C is null-homotopic (Fact 2.1).

The sequence of paths $\{u_1 u_2 ... u_r\}$ with endpoints on $bd(P)$ are detected through pointers as we traverse C on T. The homotopic path u'_i of u_i is identified by the endpoints v_1, v_2 of u_i. We do not traverse the path u'_i on $bd(P)$ which can have $\Omega(n)$ length. To form the word $w_i \sim u_i$ we detect the corresponding vertices of v_1, v_2 on $bd(Q)$ through pointers from P to Q. We can consider the labeled edges between these vertices on $bd(Q)$ in any direction (clockwise or counter-clockwise) to form the word w_i. This takes $O(|w_i|)$ time. Thus the word w for C is constructed in $O(|w|)$ time. The length of w can be at most $O(gk)$ since each word w_i can have at most $2\ell = O(g)$ length and there can be $r = O(k)$ such words in the worst case.

To determine if C is null-homotopic, we check if $w = 1$. The polygon Q can be thought of as a polygonal schema for $\mathcal{M}$, where 2ℓ vertices represent the same point on $\mathcal{M}$. So, w can be thought of as a word formed by the concatenation of edges taken from a canonical set of generators of $\mathcal{M}$. Detecting $w = 1$ for such a word w is known as the *word problem.* Schipper gave an algorithm for this problem which runs in $O(g|w|)$ time. Using this algorithm we can detect if $w = 1$ in $O(g^2 k)$ time; however, we can do better. If $w = w_1 w_2 ... w_r$

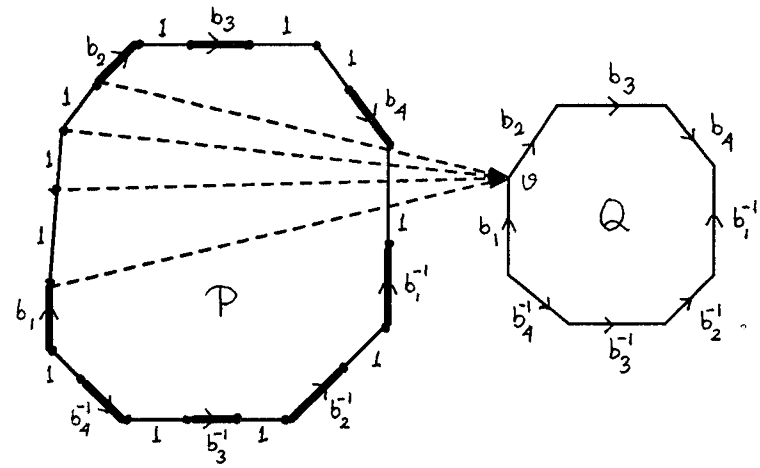


Figure 4.2: Reduced polygonal schema

where $w_i \sim u_i$, we can detect if $w = 1$ in $O(gr) = O(gk)$ time.

Our method is similar to the method of Schipper [6]. Let $\mathcal{U}(\mathcal{M})$ denote the universal covering space for $\mathcal{M}$. We can construct a universal covering space $\mathcal{U}(\mathcal{M})$ of $\mathcal{M}$ as follows: Let $P_{\mathcal{M}}$ be a $4g$-gon (if $\mathcal{M}$ is orientable) or a $2g$-gon (if $\mathcal{M}$ is non-orientable) such that when its edges are identified we get a set of generators for $\mathcal{M}$ meeting at a single point. By taking infinitely many copies of $P_{\mathcal{M}}$ and gluing them together along the identified edges, one gets a tessellation of the plane with either $4g$-gons or $2g$-gons, depending on whether $\mathcal{M}$ is orientable or not. Since the polygon Q can be thought of as a polygonal schema for $\mathcal{M}$ where all 2ℓ vertices represent the same point on $\mathcal{M}$, we can take $P_{\mathcal{M}} = Q$ to construct $\mathcal{U}(\mathcal{M})$. The following fact is well-known [8].

Fact 5.1: Any curve C on $\mathcal{M}$ can be mapped to a unique path u (lifted path) in $\mathcal{U}(\mathcal{M})$ such that C is null-homotopic if and only if u is closed.

We construct a part of $\mathcal{U}(M)$ in an incremental fashion which is sufficient to detect if the lifted path u of w is closed. If u is closed, this method provides a closed disk whose boundary is u. For each w_i, the edge with which it starts is called its *first edge*, and the edge with which it terminates is called its *last edge*. All other edges are called *middle edges*. The first edge and the last edge of a word w_i are same if $|w_i| = 1$. We update the constructed structure when we traverse any of the first and last edges. Since all middle edges in between a first edge and a last edge are visited on a single polygon (*current polygon*), we do not need to update the structure during their visits. Because of its similarity to [6] and shortage of space, we omit the details of updation here. An improved version of this procedure is reported in [1].

An amortized analysis shows that each first and last edge take $O(g)$ amortized processing time. Since there are $O(k)$ such edges, $O(gk)$ time is spent on them altogether. Each middle edge is traversed in $O(1)$ time taking $O(gk)$ time altogether since there are $O(gk)$ such edges. Thus the total time spent on detecting the closedness of the lifted path for w is $O(gk)$. Unfortunately, when $g \leq 1$ ($g \leq 2$ if $\mathcal{M}$ is non-orientable) the above amortized analysis fails. Indeed, for $g = 1$ ($g = 2$ if $\mathcal{M}$ is non-orientable), the above procedure takes $\Omega(k^2)$ time in total. We use different procedures for these cases as detailed below.

5.1 Special Cases

Any cycle on a sphere ($g = 0$) is contractable to a single point. So, the contractability problem is trivial for spheres. The fundamental group of any torus ($g = 1$) has the presentation $< a, b : aba^{-1}b^{-1} >$ which means $ab = ba$. Thus any word w on the canonical generators can be expressed as $w = a^m b^n$. This implies $w = 1$ if and only if $m = 0$ and $n = 0$. Let an edge e on Q (a rectangle) be traversed c_1 times in the clockwise direction and c_2 times in the counter-clockwise direction on the paths/words $w_1, w_2, ..., w_r$. Let $c^e = c_1 - c_2$. We have $m = 0$ and $n = 0$ if and only if $c^e + c^{e'} = 0$ for each pair of partnered edges e, e'. Detecting this fact takes $O(\sum_{i=1}^{r} |w_i|) = O(gk) = O(k)$ time. With the $O(n)$ preprocessing time we have an $O(n + k)$ time algorithm for torus.

Similar to the orientable 2-manifolds, the cases when $g = 1$ (projective planes) and $g = 2$ (klein bottle) for non-orientable 2-manifolds are treated separately. Although the case for $g = 1$ can be solved trivially, the case for $g = 2$ (polygon Q is rectangle) cannot be solved by the method used for torus. Instead we use a different method for klein bottles.

For klein bottles, the polygon Q is a rectangle and the sequence of edges around $bd(Q)$ reads either $abab^{-1}$ or $aabb$; see Figure 5.1. The universal covering space with these two patterns are shown in Figure 5.1. We observe that vertices in the universal covering space have two

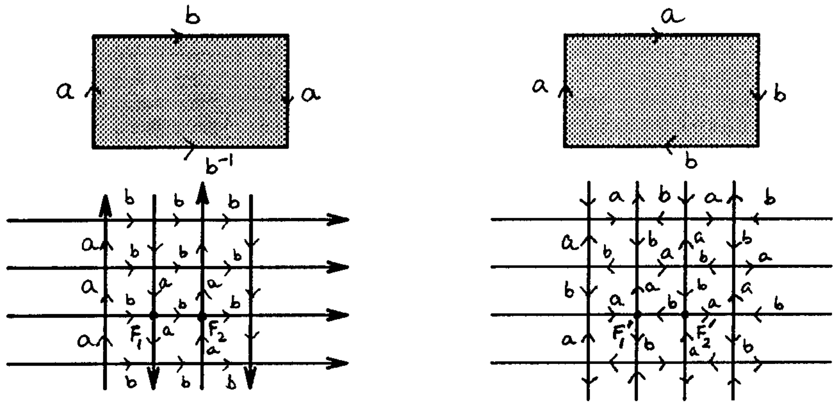


Figure 5.1: Universal covering spaces for klein bottles

different configurations in both cases. We can determine in an $O(1)$ step the edges that take us left, right, up and down for each configuration. For example, from configuration F_1 (Figure 5.1) we go left, right, up and down by the edges b^{-1}, b, a^{-1} and a respectively. We can also decide in $O(1)$ time which configuration we end up with when we move from a particular configuration through a particular edge. For example, we go to F_2' from F_1' through the edge b^{-1} (Figure 5.1). Once we have this information available through a preprocessing step, we can visit the edges of w on $bd(Q)$ and for each such edge we know the direction of our movement in the universal covering space without constructing it explicitly. The path on the universal covering space will be closed if and only if we move an equal number of times left as right and an equal number of times up as down. This procedure takes $O(|w|) = O(gk) = O(k)$ time. With preprocessing times added, we have an optimal $O(n + k)$ time and space algorithm.

Combining the costs of all steps for all cases we have the following theorem.

Theorem 5.1 Let $\mathcal{M}$ be any compact 2-manifold without boundary of genus $g \geq 0$ with a triangulation T of size n. We can detect if a given cycle C of length k is null-homotopic in $O(n + gk)$ time and space.

5.2 Related Problems

Detecting homotopic cycles: Two cycles C_1, C_2 of lengths k_1, k_2 respectively are homotopic if and only if $[C_1 \cdot C_2^{-1}] = 1$. Let w_1, w_2 be the words constructed corresponding to C_1 and C_2^{-1} respectively. We have $[C_1 \cdot C_2^{-1}] = 1$ if and only if $w_1.w_2^{-1} = 1$. This can be detected in $O(g(k_1 + k_2))$ time giving an $O(n + g(k_1 + k_2))$ time algorithm for this problem.

Computing actual contractions: The path u corresponding to w on the partially constructed universal space is closed if and only if the original cycle C is null-homotopic. In case u is closed we can compute the actual contraction as follows. Recall that the closed disk whose boundary is u is present in the constructed structure. First, the cycle C is deformed to the cycle $C' = u_1'u_2'...u_r'$ on G. This is achieved by deforming $u_1, u_2, ..., u_r$ to $u_1', u_2', ..., u_r'$ respectively over the triangles of T in $O(kn)$ time. Next we contract all the edges of G that are present in the spanning tree Y in $O(n)$ time. This may leave some of the triangles of T (equivalently T' of P) to be contracted to a single edge loop or a double edge loop. These two steps deform the cycle C to u, which is contracted over the polygons in the constructed structure. The deformed triangulation T' is superimposed on each such polygon Q. A contraction of u over the deformed triangulations of the polygons completes the contraction of C. The constructed partial structure can have at most $O(gk)$ polygons since it is constructed in $O(gk)$ time. This implies that u can be contracted over $O(gk)$ triangulations of size $O(n)$ each. This produces an $O(gkn)$ time and space actual contraction for C.

Remark: Recently using our polygonal schema computation and a clever modification of Schipper's universal covering space construction with the help of sophisticated data structures, we have designed an improved algorithm for the null homotopy problem [1]. This algorithm runs in $O(n+k\sqrt{\log g})$ time and in $\Theta(n+k)$ space. The actual contraction can be computed in $\Theta(nk)$ time and space.

6 Conclusions

We have presented a new technique to derive a reduced polygonal schema from a triangulation of a given 2-manifold. This has produced an improved algorithm for detecting null-homotopic cycles on compact 2-manifolds without boundary. We believe that this technique will find further applications in other algorithms on 2-manifolds. The method can possibly be adapted to solve the contractability problem on compact 2-manifolds with boundary. An open question is: can the recent time bound $O(n + k\sqrt{\log g})$ achieved for null-homotopy problem be further improved to $\Theta(n + k)$?

An extension of the null-homotopy problem is to consider it on arbitrary complexes. Fundamental group of any d-dimensional ($d > 1$) complex is determined only by its 2 dimensional subcomplex. So, to solve the contractability problem we can concentrate on only 2-complexes. However, it is known that the contractability problem is unsolvable for arbitrary 4-manifolds [8]. Hence the contractability problem is unsolvable for arbitrary 2-complexes. It is an interesting and probably a difficult question to characterize the 2-complexes for which the contractability problem is solvable. Perhaps a more practical problem would be to consider the 2-complexes that are embeddable in three dimensions.

Acknowledgement. The author wishes to thank Herbert Edelsbrunner for sharing an idea on the special case of klein bottles.

References

[1] T. K. Dey and H. Schipper. A new technique to compute polygonal schema for 2-manifolds with application to null-homotopy detection. *manuscript* (1994).

[2] H. Edelsbrunner and C. Delfinado. An incremental algorithm for betti numbers of simplicial complexes. *9th. Ann. Sympos. Comput. Geom.* (1993), 232–239.

[3] P. J. Giblin. Graphs, Surfaces and Homology. *Halsted Press.* New York (1977).

[4] J. Hersberger and J. Snoeyink. Around and around: computing the shortest loop. *Proc. 3rd Canad. Conf. on Comput. Geom.* (1991), 157–161.

[5] K. Melhorn and C. K. Yap. Constructive Hopf's theorem: or how to untangle closed planar curves. *Proc. ICALP, Springer Lecture Notes in Computer Science.* (1988), 410-423.

[6] H. Schipper. Determining contractibility of curves. *Proc. 8th ACM Sympos. on Comput. Geom.* (1992), 358–367.

[7] J. S. Snoeyink. Topological approaches in computational geometry. *Tech. Report STAN-CS-90-1331* (1990).

[8] J. Stillwell. Classical topology and combinatorial group theory. *Springer-Verlag.* New York (1980).

[9] G. Vegter. Kink-free deformations of polygons. *Proc. 5th ACM Sympos. on Comput. Geom.* (1989), 61–68.

[10] G. Vegter and C. K. Yap. Computational complexity of combinatorial surfaces. *Proc. 6th ACM Sympos. on Comput. Geom.* (1990), 102–111.

Triangulating Topological Spaces*

Herbert Edelsbrunner and Nimish R. Shah
Department of Computer Science
University of Illinois, Urbana, IL 61801, USA.

Abstract

Given a subspace $\mathbf{X} \subseteq \mathbb{R}^d$ and a finite set $S \subseteq \mathbb{R}^d$, we introduce the Delaunay simplicial complex, $\mathcal{D}_{\mathbf{X}}$, restricted by $\mathbf{X}$. Its simplices are spanned by subsets $T \subseteq S$ for which the common intersection of Voronoi cells meets $\mathbf{X}$ in a non-empty set. By the nerve theorem, $\bigcup \mathcal{D}_{\mathbf{X}}$ and $\mathbf{X}$ are homotopy equivalent if all such sets are contractible. This paper shows that $\bigcup \mathcal{D}_{\mathbf{X}}$ and $\mathbf{X}$ are homeomorphic if the sets can be further subdivided in a certain way so they form a regular CW complex.

1 Introduction

This paper studies the problem of constructing simplicial complexes that represent or approximate a geometric object in some finite-dimensional Euclidean space, $\mathbb{R}^d$. We refer to the geometric object as a topological space or subspace of $\mathbb{R}^d$. This problem arises in geometric modeling and finite element analysis, and it is a special case of the grid generation problem [21]. It is special because we only consider grids or complexes made up of simplices. The problem can be divided into two questions:

> How do we choose the points or vertices of the grid?
>
> How do we connect the vertices using edges, triangles, and higher-dimensional simplices?

In this paper we concentrate on the second question. In particular, given a subspace and a finite point set in $\mathbb{R}^d$, we give an unambiguous rule for constructing a simplicial complex representing the subspace. Topological properties of this simplicial complex, such as whether its domain is homotopy equivalent or homeomorphic to the subspace, can be studied based on local interactions between the subspace and the Voronoi neighborhoods of the points. This leads us back to the first question: additional points can be chosen so they improve the local interaction patterns. How this can be done in a concrete, possibly three-dimensional setting ought to be the subject of future investigations.

Simplicial complexes and triangulations. An affinely independent point set $T \subseteq \mathbb{R}^d$ defines the *simplex* $\sigma_T = \operatorname{conv} T$. Its *dimension* is $k = \dim \sigma_T = \operatorname{card} T - 1$, and it is also referred to as a *k-simplex*. The points of T are the *vertices* of σ_T. A *simplicial complex*, $\mathcal{K}$, is a finite collection of simplices that satisfies the following two properties: if $\sigma_T \in \mathcal{K}$ and $U \subseteq T$ then $\sigma_U \in \mathcal{K}$, and if $\sigma_T, \sigma_V \in \mathcal{K}$ then $\sigma_T \cap \sigma_V = \sigma_{T \cap V}$. The first property implies $\emptyset \in \mathcal{K}$, and the first and second properties combined imply $\sigma_T \cap \sigma_V \in \mathcal{K}$. The *vertex set* of $\mathcal{K}$ is $\operatorname{vert} \mathcal{K} = \bigcup_{\sigma_T \in \mathcal{K}} T$, the *dimension* is $\dim \mathcal{K} = \max_{\sigma \in \mathcal{K}} \dim \sigma$, and the *underlying space* is $\bigcup \mathcal{K} = \bigcup_{\sigma \in \mathcal{K}} \sigma$. A *subcomplex* of $\mathcal{K}$ is a simplicial complex $\mathcal{L} \subseteq \mathcal{K}$.

A particular simplicial complex defined by a non-degenerate finite set $S \subseteq \mathbb{R}^d$ is the *Delaunay simplicial complex*, $\mathcal{D} = \mathcal{D}_S$ [6]. It consists of all simplices σ_T, $T \subseteq S$, for which there exists an open ball, B, with $S \cap \operatorname{cl} B = T$ and $S \cap B = \emptyset$. Given S, $\mathcal{D}$ is unique, $\dim \mathcal{D} = \min\{d, \operatorname{card} S - 1\}$, and $\bigcup \mathcal{D} = \operatorname{conv} S$. In computational geometry, $\mathcal{D}$ is usually referred to as the Delaunay 'triangulation' of S [7, 18]. To avoid confusion with the topology notion of a triangulation, which is adopted in this paper, we choose to call $\mathcal{D}$ a simplicial complex. Following the tradition in combinatorial topology, a *triangulation* of a topological space $\mathbf{X}$ is a simplicial complex $\mathcal{K}$ together with a homeomorphism between $\mathbf{X}$ and $\bigcup \mathcal{K}$ [17, 19]. If there exists a simplicial complex $\mathcal{K}$ such that $\bigcup \mathcal{K}$ is homeomorphic to $\mathbf{X}$, then $\mathbf{X}$ is *triangulable*.

*Research of both authors was supported by the National Science Foundation under grant ASC-9200301 and under the Alan T. Waterman award, grant CCR-9118874. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the National Science Foundation.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

Outline. Our approach to constructing a simplicial complex that represents a topological space $\mathbb{X} \subseteq \mathbb{R}^d$ is based on a finite set $S \subseteq \mathbb{R}^d$ and the Delaunay simplicial complex of this set, $\mathcal{D} = \mathcal{D}_S$. Section 2 introduces the concept of a Delaunay simplicial complex $\mathcal{D}_{\mathbb{X}}$ restricted by $\mathbb{X}$, which is a subcomplex of $\mathcal{D}$. This concept is a common generalization of ideas developed by Martinetz and Schulten [15], Chew [4], and Edelsbrunner [8]. $\mathcal{D}_{\mathbb{X}}$ is defined for every $\mathbb{X} \subseteq \mathbb{R}^d$ and every non-degenerate finite $S \subseteq \mathbb{R}^d$. If S satisfies the assumptions of the nerve theorem, see section 2, then $\bigcup \mathcal{D}_{\mathbb{X}}$ and $\mathbb{X}$ are homotopy equivalent. Section 3 presents some topological concepts and discusses the meaning of non-degeneracy in detail. Sections 4 and 5 study conditions on S that guarantee $\bigcup \mathcal{D}_{\mathbb{X}}$ be homeomorphic to $\mathbb{X}$. An explicit construction of a homeomorphism is also given. Section 6 mentions directions for further research.

2 Restricted Delaunay Simplicial Complexes

Coverings and nerves. Let $\mathbb{X} \subseteq \mathbb{R}^d$ be a topological space and $S \subseteq \mathbb{R}^d$ a finite point set. We assume non-degenerate position of the points in S, which generically means that anything vanishing under a slight perturbation of the points is precluded. For example, we require that $T \subseteq S$ be affinely independent if $\operatorname{card} T \leq d+1$, and that no $d+2$ points of S be cospherical. Further particulars of this assumption will be discussed in section 3. The *Voronoi cell* of $p \in S$ is

$$V_p = \{x \in \mathbb{R}^d \mid |px| \leq |qx|,\ q \in S\},$$

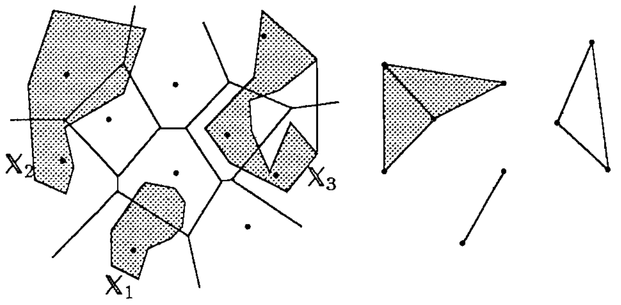
where $|yz|$ denotes the Euclidean distance between points $y, z \in \mathbb{R}^d$. The collection of Voronoi cells is $\mathsf{V} = \mathsf{V}_S = \{V_p \mid p \in S\}$ [22]. The *Voronoi cell restricted to* $\mathbb{X}$ of $p \in S$ is $V_{p,\mathbb{X}} = \mathbb{X} \cap V_p$, and the collection of restricted Voronoi cells is $\mathsf{V}_{\mathbb{X}} = \mathsf{V}_{S,\mathbb{X}} = \{V_{p,\mathbb{X}} \mid p \in S\}$. For a subset $T \subseteq S$ we have corresponding subsets $\mathsf{V}_T = \{V_p \mid p \in T\} \subseteq \mathsf{V}$ and $\mathsf{V}_{T,\mathbb{X}} = \{V_{p,\mathbb{X}} \mid p \in T\} \subseteq \mathsf{V}_{\mathbb{X}}$. We will consider their common intersections, $\bigcap \mathsf{V}_T$ and $\bigcap \mathsf{V}_{T,\mathbb{X}} = \mathbb{X} \cap \bigcap \mathsf{V}_T$.

A *covering* of $\mathbb{X}$ is a collection C of subsets of $\mathbb{X}$ so that $\mathbb{X} = \bigcup \mathsf{C}$. It is a *closed (open)* covering if each set in C is closed (open), and it is a *finite* covering if C is finite. For a subset $\mathsf{D} \subseteq \mathsf{C}$ consider the common intersection, $\bigcap \mathsf{D}$. The *nerve* of a finite covering C is

$$\operatorname{nerve} \mathsf{C} = \{\mathsf{D} \subseteq \mathsf{C} \mid \bigcap \mathsf{D} \neq \emptyset\}.$$

We remark that the nerve can be defined for a finite covering of any abstract set, not just for subsets of $\mathbb{R}^d$. A *geometric realization* of $\operatorname{nerve} \mathsf{C}$ is a simplicial complex, $\mathcal{K}$, together with a bijection β between C and the vertex set of $\mathcal{K}$, so that $\mathsf{D} \in \operatorname{nerve} \mathsf{C}$ iff the simplex spanned by $\beta(\mathsf{D})$ is in $\mathcal{K}$.

Observe that the collection of Voronoi cells restricted to $\mathbb{X}$, $\mathsf{V}_{\mathbb{X}}$, is a finite closed covering of $\mathbb{X}$. The *Delaunay simplicial complex restricted* by $\mathbb{X}$, $\mathcal{D}_{\mathbb{X}} = \mathcal{D}_{S,\mathbb{X}}$, is the geometric realization of $\operatorname{nerve} \mathsf{V}_{\mathbb{X}}$ defined by $\beta(V_p) = p$, for all $p \in S$. That is, $\mathcal{D}_{\mathbb{X}} = \{\sigma_T \mid T \subseteq S,\ \bigcap \mathsf{V}_{T,\mathbb{X}} \neq \emptyset\}$. Note that $\mathcal{D}_{\mathbb{X}}$ is a subcomplex of the Delaunay simplicial complex $\mathcal{D} = \mathcal{D}_{\mathbb{R}^d}$ of S. See figure 2.1 for an example. The nerve theorem of combinatorial topology [2, 13, 23] sheds some light on the relationship between $\mathbb{X}$, $\mathsf{V}_{\mathbb{X}}$, and $\mathcal{D}_{\mathbb{X}}$. See section 3 for a formal definition of homotopy equivalence and contractibility.



(a) (b)

Figure 2.1: (a) The Voronoi cells of ten points decompose three spaces, $\mathbb{X}_1$, $\mathbb{X}_2$, and $\mathbb{X}_3$. (b) The three corresponding Delaunay simplicial complexes consist of an edge, a cycle of three edges, and two triangles sharing an edge, respectively.

THEOREM (nerve). Let C be a finite closed covering of a triangulable space $\mathbb{X} \subseteq \mathbb{R}^d$ so that for every $\mathsf{D} \subseteq \mathsf{C}$, $\bigcap \mathsf{D}$ is either empty or contractible. Let $\mathcal{K}$ be a geometric realization of $\operatorname{nerve} \mathsf{C}$. Then $\mathbb{X}$ and $\bigcup \mathcal{K}$ are homotopy equivalent.

In particular, if $\bigcap \mathsf{D}$ is empty or contractible for every $\mathsf{D} \subseteq \mathsf{V}_{\mathbb{X}}$ then $\mathbb{X}$ and $\bigcup \mathcal{D}_{\mathbb{X}}$ are homotopy equivalent.

Related earlier work. Martinetz and Schulten [15] study neural nets modeling a topological space $\mathbb{X}$. In geometric language, a *neural net* consists of finitely many points and edges between them. The algorithm in [15] constructs the net by choosing a finite set $S \subseteq \mathbb{X}$ and selecting edges based on points sampled from $\mathbb{X}$ found in Voronoi cells associated with point pairs. A cell is associated with every pair $\{p, q\} \subseteq S$ for which $V_p \cap V_q \neq \emptyset$, see [7, 18]. This cell can be interpreted as a "thickened" version of the face common to V_p and V_q, and a point of $\mathbb{X}$ sampled in this cell is taken as evidence that $\mathbb{X} \cap V_p \cap V_q \neq \emptyset$. Martinetz and Schulten call the resulting net the *induced* Delaunay triangulation of S and $\mathbb{X}$; in the limit it is the same as the edge-skeleton of $\mathcal{D}_{\mathbb{X}}$.

Chew [4] introduces a method for constructing a simplicial complex approximating a (two-dimensional) surface in $\mathbb{R}^3$. Let $\mathbb{X}$ be such a surface and $S \subseteq \mathbb{X}$ a finite point set. Three points $p, q, r \in S$ span a triangle in the approximating complex if they lie on the boundary of an open ball $B \subseteq \mathbb{R}^3$ with center in $\mathbb{X}$ so that $S \cap B = \emptyset$. Assuming non-degeneracy, such a ball B exists iff $\mathbb{X} \cap V_p \cap V_q \cap V_r \neq \emptyset$. Chew's method can thus be seen as a special case of our definition of restricted Delaunay simplicial complexes.

Finally, Edelsbrunner [8] defines the dual complex of a union of balls in $\mathbb{R}^d$. Consider the case where all balls are equally large. Let $S \subseteq \mathbb{R}^d$ be a finite point set, and define $\mathbb{X} = \{x \in \mathbb{R}^d \mid \min_{p \in S} |xp| \leq \rho\}$, for some fixed positive $\rho \in \mathbb{R}$. The Voronoi cells of S decompose $\mathbb{X}$ into closed convex regions, and the dual complex is defined as the nerve of these regions, geometrically realized by the map $\beta(V_{p,\mathbb{X}}) = p$, for all $p \in S$. We see that it is the same as the Delaunay simplicial complex, $\mathcal{D}_\mathbb{X}$, restricted by $\mathbb{X}$. The common intersection of any subset of these regions is convex and therefore contractible, so the nerve theorem implies that the underlying space of the dual complex is homotopy equivalent to $\mathbb{X}$.

3 Some Topological Concepts

Neighborhoods, homotopies, homeomorphisms, and manifolds. An *open ball* in $\mathbb{R}^d$ is a set $B = B(x, \rho) = \{y \in \mathbb{R}^d \mid |yx| < \rho\}$ for some point $x \in \mathbb{R}^d$ and some positive $\rho \in \mathbb{R}$; x is the *center* and ρ the *radius* of B. For $\mathbb{Y} \subseteq \mathbb{X}$, a *neighborhood* of $\mathbb{Y}$ in $\mathbb{X}$ is an open subset of $\mathbb{X}$ that contains $\mathbb{Y}$.

Let $\mathbb{X}$ and $\mathbb{Y}$ be two topological spaces. Two maps $f, g : \mathbb{X} \to \mathbb{Y}$ are *homotopic* if there is a continuous map $h : \mathbb{X} \times [0,1] \to \mathbb{Y}$ with $h(x, 0) = f(x)$ and $h(x, 1) = g(x)$ for all $x \in \mathbb{X}$. The two spaces, $\mathbb{X}$ and $\mathbb{Y}$, are *homotopy equivalent* if there are continuous maps $f : \mathbb{X} \to \mathbb{Y}$ and $g : \mathbb{Y} \to \mathbb{X}$ so that $g \circ f$ is homotopic to the identity map in $\mathbb{X}$ and $f \circ g$ is homotopic to the identity map in $\mathbb{Y}$. $\mathbb{X}$ is *contractible* if it is homotopy equivalent to a point.

Topological spaces $\mathbb{X}$ and $\mathbb{Y}$ are *homeomorphic*, written $\mathbb{X} \approx \mathbb{Y}$, if there is a bijective map $\varphi : \mathbb{X} \to \mathbb{Y}$ so that φ and φ^{-1} are continuous. φ is a *homeomorphism* between $\mathbb{X}$ and $\mathbb{Y}$, and $\mathbb{X}$ and $\mathbb{Y}$ are *homeomorphs* of each other. Homeomorphs of open, half-open, and closed balls of various dimensions play an important role in the forthcoming discussions. For $k \geq 0$, let o be the origin of $\mathbb{R}^k$ and define

$$\begin{aligned} \mathbb{H}^k &= \{x = (\xi_1, \ldots, \xi_k) \in \mathbb{R}^k \mid \xi_k \geq 0\}, \\ \mathbb{B}^k &= \{x \in \mathbb{R}^k \mid |x\mathsf{o}| \leq 1\}, \quad \text{and} \\ \mathbb{S}^{k-1} &= \{x \in \mathbb{R}^k \mid |x\mathsf{o}| = 1\}. \end{aligned}$$

For convenience, we define $\mathbb{R}^k = \mathbb{H}^k = \mathbb{B}^k = \mathbb{S}^k = \emptyset$ if $k < 0$. An *open k-ball* is a homeomorph of $\mathbb{R}^k$, a *half-open k-ball* is a homeomorph of $\mathbb{H}^k$, a *closed k-ball* is a homeomorph of $\mathbb{B}^k$, and a $(k-1)$-*sphere* is a homeomorph of $\mathbb{S}^{k-1}$. For $k \geq 1$ these are disjoint classes of spaces, that is, open balls, half-open balls, closed balls, and spheres are pairwise non-homeomorphic. This is not true for $k = 0$: open, half-open, and closed 0-balls are points, and a 0-sphere is a pair of points.

Our first theorem is about topological spaces that are manifolds, with or without boundary. $\mathbb{X} \subseteq \mathbb{R}^d$ is a *k-manifold without boundary* if each $x \in \mathbb{X}$ has an open k-ball as a neighborhood in $\mathbb{X}$. $\mathbb{X} \subseteq \mathbb{R}^d$ is a *k-manifold with boundary* if each $x \in \mathbb{X}$ has an open or half-open k-ball as a neighborhood in $\mathbb{X}$, and there is at least one $x \in \mathbb{X}$ that has no open k-ball as a neighborhood. The set of points without open k-ball neighborhood forms the *boundary*, $\mathrm{bd}\,\mathbb{X}$, of $\mathbb{X}$. Note that the boundary of a half-open k-ball is an open $(k-1)$-ball, which is therefore without boundary. From this it follows that the boundary of a k-manifold with boundary is a $(k-1)$-manifold without boundary. The *interior* of a manifold $\mathbb{X}$ is $\mathrm{int}\,\mathbb{X} = \mathbb{X} - \mathrm{bd}\,\mathbb{X}$; it is the set of points with open k-ball neighborhoods. Note that our definition distinguishes between manifolds with and without boundary, which is somewhat non-standard as the set of manifolds without boundary is usually considered a subset of the set of manifolds with boundary. A manifold $\mathbb{X}$ is *compact* if every open covering of $\mathbb{X}$ has a finite subcovering, or equivalently, if it is closed and bounded. A manifold $\mathbb{Y} \subseteq \mathbb{X}$ is a *submanifold* of $\mathbb{X}$.

Non-degeneracy. In section 4, we are interested in the intersection between manifolds and affine flats. An ℓ-manifold $F \subseteq \mathbb{R}^d$ is an *ℓ-flat* if it is the affine hull of $\ell+1$ points, or equivalently, it is the intersection of $d-\ell$ hyperplanes, or linear functionals. Let $\mathbb{X} \subseteq \mathbb{R}^d$ be an m-manifold. Intuitively, a point $x \in \mathbb{R}^d$ has d degrees of freedom, and it looses $d-m$ of them if it is constrained to lie in $\mathbb{X}$. Similarly, x looses $d-\ell$ degrees if it is constrained to lie in F. So if $x \in \mathbb{X} \cap F$ then x should have lost $2d-m-\ell$ degrees of freedom. Hence, $\mathbb{X} \cap F$ should be empty if $d-(2d-m-\ell) = m+\ell-d < 0$. In general, we expect x to have $m+\ell-d$ degrees of freedom and $\mathbb{X} \cap F$ to be an $(m+\ell-d)$-manifold. Algorithmically, such a non-degeneracy assumption can be simulated by conceptual perturbation techniques, see e.g. [9, 24].

This intuitive argument can be formalized for smooth or piecewise smooth manifolds. We need some definitions from differential topology. For an open set $\mathbb{X} \subseteq \mathbb{R}^m$, a map $f : \mathbb{X} \to \mathbb{R}^n$ is *smooth* if it has continuous partial derivatives of all orders and at all points of $\mathbb{X}$. For arbitrary $\mathbb{X} \subseteq \mathbb{R}^m$, f is *smooth* if for all $x \in \mathbb{X}$ there exists an open ball $B = B(x, \varepsilon) \subseteq \mathbb{R}^m$ and a smooth map $g : B \to \mathbb{R}^n$ so that g equals f on $\mathbb{X} \cap B$.

Topological spaces $\mathbb{X} \subseteq \mathbb{R}^m$ and $\mathbb{Y} \subseteq \mathbb{R}^n$ are *diffeomorphic* if there exists a homeomorphism $\varphi : \mathbb{X} \to \mathbb{Y}$ so that φ and φ^{-1} are smooth. An m-manifold $\mathbb{X}$ with or without boundary is *smooth* if each $x \in \mathbb{X}$ has a neighborhood diffeomorphic to $\mathbb{R}^k$ or $\mathbb{H}^k$.

Now, let $\mathbb{X}$ be a smooth m-manifold, and let $f : \mathbb{X} \to \mathbb{R}$ be a smooth map. Then $y \in \mathbb{R}$ is a *regular value* of f if for every $x \in f^{-1}(y)$ some partial derivative of f at x is non-zero; otherwise, y is a *critical value* of f. By the preimage theorem in differential topology, the preimage of any regular value is a smooth submanifold of $\mathbb{X}$ with dimension $m-1$, and by Sard's theorem, the set of critical values has measure 0 in $\mathbb{R}$, see e.g. [10, 11]. This implies that with probability 1, the intersection between a smooth m-manifold $\mathbb{X}$ and a hyperplane with prescribed normal direction is a smooth $(m-1)$-manifold. Hence, with probability 1, the intersection between $\mathbb{X}$ and an ℓ-flat F with prescribed normal $(d-\ell)$-flat is a smooth $(m+\ell-d)$-manifold. By non-degenerate position of F we mean that this is indeed the case, and it is reasonable to assume non-degenerate position because F just needs to avoid a measure zero set in $\mathbb{R}^{d-\ell}$.

One of the conditions necessary for our results is the non-degeneracy of the intersection between Voronoi cells and the manifold. We thus extend the above notions to Voronoi cells and their intersections. An intersection of Voronoi cells is the common intersection of finitely many closed half-spaces, and thus a *convex polyhedron.* Let P be a convex polyhedron and let $\mathbb{X}$ be a manifold without boundary. We say that P *intersects* $\mathbb{X}$ *generically* if $\mathbb{X} \cap \mathsf{P} = \emptyset$ or $\mathbb{X} \cap \mathsf{P}$ has the right dimension and $\mathbb{X} \cap \operatorname{int} \mathsf{P} = \operatorname{int}(\mathbb{X} \cap \mathsf{P})$. If $\mathbb{X}$ is a manifold with boundary, then P *intersects* $\mathbb{X}$ *generically* if P intersects $\operatorname{int} \mathbb{X}$ and $\operatorname{bd} \mathbb{X}$ generically. By non-degenerate position of P we mean that P intersects $\mathbb{X}$ generically. Again, this is a reasonable assumption to make.

4 Triangulating Compact Manifolds

We are now ready to state conditions under which the restricted Delaunay simplicial complex is a triangulation of $\mathbb{X}$. These conditions will be applied only when $\mathbb{X}$ is a compact manifold. To avoid any confusion, we note that our results do not settle the open question whether or not all compact manifolds are triangulable.

The closed ball property. Throughout this section we assume non-degenerate position of flats and discrete point sets. Let $m > 0$ and let $\mathbb{X} \subseteq \mathbb{R}^d$ be a compact m-manifold with or without boundary. Let $S \subseteq \mathbb{R}^d$ be a finite point set. We say that S has the *generic intersection property* for $\mathbb{X}$ if for every subset $T \subseteq S$, $\bigcap \mathsf{V}_T$ intersects $\mathbb{X}$ generically. We say that S has the *closed ball property* for $\mathbb{X}$ if for every $\ell \leq m$ and every subset $T \subseteq S$ with $\operatorname{card} T = m + 1 - \ell$, the following two conditions hold:

(B1) $\bigcap \mathsf{V}_{T,\mathbb{X}}$ is either empty or a closed ℓ-ball, and

(B2) $\bigcap \mathsf{V}_{T,\operatorname{bd}\mathbb{X}}$ is either empty or a closed $(\ell-1)$-ball.

In section 3, we argued that it is reasonable to expect $\mathbb{X} \cap \bigcap \mathsf{V}_T$ be an ℓ-manifold, if $\mathbb{X}$ is a smooth m-manifold. This is reflected in condition (B1). A similar non-degenerate position assumption is implied by (B2) for the boundary of $\mathbb{X}$. We will see that the closed ball property guarantees that dimension is preserved. As an example consider $\mathbb{X}_1$ and $\mathcal{D}_{\mathbb{X}_1}$ in figure 2.1. Condition (B2) is violated by the common edge of the two Voronoi cells intersecting $\mathbb{X}_1$. Indeed, $\bigcup \mathcal{D}_{\mathbb{X}_1} \not\approx \mathbb{X}_1$ because the dimension of $\mathbb{X}_1$ is two and that of $\bigcup \mathcal{D}_{\mathbb{X}_1}$ is one.

Two technical lemmas. Before proving the firts theorem we establish two facts about the closed ball property. The first says that the closed ball property preserves dimension locally, and the second is a statement about the way Voronoi cells intersect a compact manifold and its boundary. We work with arbitrary, that is, possibly non-smooth m-manifolds since we do not need any smoothness properties for our proofs. A simplex $\sigma_T \in \mathcal{D}_{\mathbb{X}}$ is a *principal simplex* if there is no proper superset $U \supset T$ with $\sigma_U \in \mathcal{D}_{\mathbb{X}}$. The first fact is formalized in the following lemma.

LEMMA 4.1 (preservation of dimension). Let $m \geq 1$, let $\mathbb{X} \subseteq \mathbb{R}^d$ be a compact m-manifold with or without boundary, and let $S \subseteq \mathbb{R}^d$ be a non-degenerate finite set of points that has the generic intersection property for $\mathbb{X}$. If S has the closed ball property for $\mathbb{X}$ then every principal simplex of $\mathcal{D}_{\mathbb{X}}$ is an m-simplex.

PROOF. Let $\sigma_T \in \mathcal{D}_{\mathbb{X}}$ be a principal simplex, and define $F' = \bigcap \mathsf{V}_T$ and $F = \bigcap \mathsf{V}_{T,\mathbb{X}}$. By condition (B1), $F = \mathbb{X} \cap F'$ is a closed ℓ-ball, with $\ell = m + 1 - \operatorname{card} T$. Since $\sigma_T \in \mathcal{D}_{\mathbb{X}}$, we have $F \neq \emptyset$, which implies $\ell \geq 0$ and therefore $\operatorname{card} T \leq m{+}1$. If $\operatorname{card} T = m{+}1$ then σ_T is an m-simplex and we are done. So suppose $\operatorname{card} T < m{+}1$. Since σ_T is a principal simplex, it follows that $F \subseteq \operatorname{int} F'$, for otherwise there is a proper face $G' = \bigcap \mathsf{V}_U$ of F', so $T \subset U$, with $\mathbb{X} \cap G' \neq \emptyset$. If follows that $\sigma_U \in \mathcal{D}_{\mathbb{X}}$, which contradicts the principality of σ_T.

Finally, we show that $F \subseteq \operatorname{int} F'$ also leads to a contradiction. As mentioned above, F is a closed ℓ-ball and hence $\operatorname{bd} F$ is an $(\ell-1)$-sphere. This $(\ell-1)$-sphere is contained in $\operatorname{int} F'$, so it must lie in $\operatorname{bd} \mathbb{X}$. Indeed, $\operatorname{bd} F = \operatorname{bd} \mathbb{X} \cap F'$. This contradicts condition (B2), which requires $\operatorname{bd} \mathbb{X} \cap F'$ be a closed $(\ell-1)$-ball. □

For the next lemma we need a classic result on subdivisions of a certain type of complex. A closed ball is called a *cell*, or a *k-cell* if its dimension is k. A finite collection of non-empty cells, $\mathcal{R}$, is a *regular CW complex* if the cells have pairwise disjoint interiors, and the boundary of each cell is the union of other cells in $\mathcal{R}$. A subset of $\mathcal{R}$ is a *chain* if its elements can be ordered so that each contains its predecessors and is contained in its successors. Let $\mathsf{C}_{\mathcal{R}}$ be the set of chains in $\mathcal{R}$ and note that nerve $\mathsf{C}_{\mathcal{R}}$ is well defined. The result mentioned is that any geometric realization of nerve $\mathsf{C}_{\mathcal{R}}$ is homeomorphic to $\bigcup \mathcal{R}$. It can be found in [5, chapter V] and [14, chapter III] and also in [3] where it is applied to manifolds subdivided by the cells of a regular CW complex. Another classic result needed is the weak Schönflies theorem, see e.g. [20, chapter 3], which implies that if A is a piecewise linear k-sphere and $B \subset A$ is a piecewise linear closed k-ball then $A - \operatorname{int} B$ is also a closed k-ball.

LEMMA 4.2 (complimentary closed ball property). Let m, $\mathbb{X}$, and S be as in lemma 4.1. Let $T \subseteq S$ be so that $G = \bigcap \mathsf{V}_{T,\operatorname{bd}\mathbb{X}} \neq \emptyset$, and define $F = \bigcap \mathsf{V}_{T,\mathbb{X}}$ and $\ell = m - \operatorname{card} T$. If S has the closed ball property for $\mathbb{X}$ then $\operatorname{bd} F - \operatorname{int} G$ is a closed ℓ-ball.

PROOF. By condition (B2), F is a closed $(\ell+1)$-ball, and thus an $(\ell+1)$-manifold with boundary. Let $\mathcal{R}$ consist of all sets $\bigcap \mathsf{V}_{U,\mathbb{X}}$ and $\bigcap \mathsf{V}_{U,\operatorname{bd}\mathbb{X}}$ over all $U \subseteq S$ with $T \subseteq U$. Assuming S has the closed ball property for $\mathbb{X}$, these sets are cells, so $\mathcal{R}$ is a regular CW complex and $F = \bigcup \mathcal{R}$. Let $\mathcal{K}$ be a geometric realization of nerve $\mathsf{C}_{\mathcal{R}}$ and let $\varphi : F \to \bigcup \mathcal{K}$ be a homeomorphism; it exists because of the homeomorphism result mentioned above. Note that $\varphi(\operatorname{bd} F) = \operatorname{bd} \bigcup \mathcal{K}$ is the underlying space of a subcomplex of $\mathcal{K}$, and similarly, $\varphi(G) = \bigcup \mathcal{L}$ for a subcomplex $\mathcal{L} \subseteq \mathcal{K}$. By construction, $\operatorname{bd} \bigcup \mathcal{K}$ is a piecewise linear ℓ-sphere and $\bigcup \mathcal{L} \subset \operatorname{bd} \bigcup \mathcal{K}$ is a piecewise linear closed ℓ-ball. The weak Schönflies theorem implies that $\operatorname{bd} \bigcup \mathcal{K} - \operatorname{int} \bigcup \mathcal{L}$ is a closed ℓ-ball. Since φ is a homeomorphism, $\varphi^{-1}(\operatorname{bd} \bigcup \mathcal{K} - \operatorname{int} \bigcup \mathcal{L}) = \operatorname{bd} F - \operatorname{int} G$ is also a closed ℓ-ball, as claimed. □

Theorem for manifolds. We need a few additional definitions. The *barycentric coordinates* of a point x with respect to a simplex σ_T, $T = \{v_0, \ldots, v_k\}$, are real numbers $\xi_0, \ldots, \xi_k$ so that $\sum_{i=0}^{k} \xi_i = 1$ and $\sum_{i=0}^{k} \xi_i v_i = x$; they are unique and non-negative if $x \in \sigma_T$. Let $\mathcal{K}$ and $\mathcal{L}$ be two simplicial complexes, and let $f : \operatorname{vert} \mathcal{K} \to \operatorname{vert} \mathcal{L}$ take the vertices of any simplex in $\mathcal{K}$ to the vertices of a simplex in $\mathcal{L}$. The *simplicial map* implied by f is $g : \bigcup \mathcal{K} \to \bigcup \mathcal{L}$, which maps a point $x \in \sigma_T$, $T = \{v_0, \ldots, v_k\}$, to $g(x) = \sum_{i=0}^{k} \xi_i f(v_i)$. We will use the fact that if f is a bijection then g is a homeomorphism. The *barycenter* of σ_T is $\mathsf{b}_T = \sum_{i=0}^{k} \frac{v_i}{k+1}$, and the *barycentric subdivision* of $\mathcal{K}$ is

$$\operatorname{sd} \mathcal{K} = \{\operatorname{conv} \{\mathsf{b}_T \mid T \in C\} \mid C \in \mathsf{C}_{\mathcal{K}}\}.$$

Figure 4.1 (c) shows the barycentric subdivision of the simplicial complex in 4.1 (b). Note that $\operatorname{sd} \mathcal{K}$ can be constructed inductively by connecting b_T to all simplices subdividing the proper faces of σ_T. The *star* of a vertex $v \in \mathcal{K}$ is $\operatorname{st} v = \{\sigma \in \mathcal{K} \mid v \in \sigma\}$.

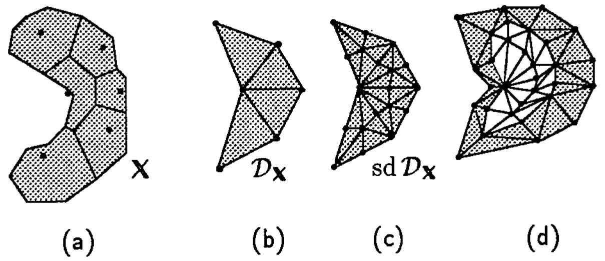


Figure 4.1: (a) The space $\mathbb{X}$ is a closed 2-ball in the form of a boomerang. It is covered by the restricted Voronoi cells of six points, which define a regular CW complex $\mathcal{R}$ with $\bigcup \mathcal{R} = \mathbb{X}$. (b) The Delaunay simplicial complex, $\mathcal{D}_{\mathbb{X}}$, consists of four triangles connecting six points. (c) The barycentric subdivision, $\operatorname{sd} \mathcal{D}_{\mathbb{X}}$, has the same underlying space as $\mathcal{D}_{\mathbb{X}}$. (d) A geometric realization of nerve $\mathsf{C}_{\mathcal{R}}$ consists of a simplicially homeomorphic copy of $\operatorname{sd} \mathcal{D}_{\mathbb{X}}$ (not shaded), surrounded by a collar of triangles (shaded).

We will use these tools to show our first result stated in theorem 4.3 below. The proof constructs a homeomorphism between $\mathbb{X}$ and $\bigcup \mathcal{D}_{\mathbb{X}}$ one step at a time. In this process, the pasting lemma of point set topology [16, 19] is employed. It can be stated as follows. If $f : A \to Y$ and $g : B \to Y$ are continuous maps that agree on $A \cap B$ and A, B are closed in $A \cup B$, then $h : A \cup B \to Y$, which agrees with f on A and with g on B is continuous.

THEOREM 4.3 Let $\mathbb{X} \subseteq \mathbb{R}^d$ be a compact manifold, with or without boundary, and let $S \subseteq \mathbb{R}^d$ be a non-degenerate finite point set that has the generic intersection property for $\mathbb{X}$. If S has the closed ball property for $\mathbb{X}$ then $\bigcup \mathcal{D}_{\mathbb{X}} \approx \mathbb{X}$.

PROOF. For each i, define $\mathsf{V}_i = \{\bigcap \mathsf{V}_{T,\mathbb{X}} \mid \operatorname{card} T = m + 1 - i\}$ and note that because of the closed ball property all elements in V_i are closed i-balls. We inductively construct simplicial complexes $\mathcal{K}_i$ and homeomorphisms $\varphi_i : \bigcup \mathsf{V}_i \to \bigcup \mathcal{K}_i$, so that $\mathcal{K}_{i-1} \subseteq \mathcal{K}_i$ and φ_i agrees with φ_{i-1} on $\bigcup \mathsf{V}_{i-1}$. When we arrive at $\mathcal{K} = \mathcal{K}_m$ and $\varphi = \varphi_m$ we show there is a simplicial homeomorphism between $\mathcal{K}$ and $\operatorname{sd} \mathcal{D}_{\mathbb{X}}$. The result follows because $\mathbb{X} = \bigcup \mathsf{V}_m \approx \bigcup \mathcal{K} \approx \bigcup \operatorname{sd} \mathcal{D}_{\mathbb{X}} = \bigcup \mathcal{D}_{\mathbb{X}}$.

To start the induction, let each $T \subseteq S$ correspond to a point v_T in $\mathbb{R}^e$. Let e be large enough and choose the points in $\mathbb{R}^e$ so that any collection of simplices of dimension up to m spanned by these points satisfy the properties of a simplicial complex. Define

$\mathcal{K}_0 = \{v_T \mid T \subseteq S,\ \operatorname{card} T = m+1,\ \bigcap \mathsf{V}_{T,\mathbb{X}} \neq \emptyset\}$. At this stage the homeomorphism $\varphi_0 : \mathsf{V}_0 \to \mathcal{K}_0$ defined by $\varphi_0(\bigcap \mathsf{V}_{T,\mathbb{X}}) = v_T$ is just a bijection between two finite point sets.

Suppose $0 \leq j \leq m-1$ and $\mathcal{K}_j$ and $\varphi_j : \bigcup \mathsf{V}_j \to \bigcup \mathcal{K}_j$ are constructed. Let $i = j+1$ and initialize $\mathcal{K}_i = \mathcal{K}_j$. Let $T \subseteq S$ with $\operatorname{card} T = m+1-i$ so that $F = \bigcap \mathsf{V}_{T,\mathbb{X}} \neq \emptyset$ and v_T is not yet in $\mathcal{K}_i$. Define $G = \bigcap \mathsf{V}_{T,\operatorname{bd}\mathbb{X}}$. We add simplices to $\mathcal{K}_i$ that will allow us to extend the homeomorphism so it includes $F \in \mathsf{V}_i$. Specifically, consider all sets $U \subseteq S$, with $T \subset U$ and $\bigcap \mathsf{V}_{U,\mathbb{X}} \neq \emptyset$. The corresponding simplices σ_U belong to $\mathcal{K}_j$ and are contained in $\varphi_j(\operatorname{bd} F - \operatorname{int} G)$. Add all simplices $\sigma_{U \cup \{v_T\}}$ to $\mathcal{K}_i$. To extend the homeomorphism we distinguish two cases. Assume first that $G = \emptyset$, and therefore $F \subseteq \operatorname{int} \mathbb{X}$. Since φ_j is a homeomorphism and F is a closed i-ball, $\operatorname{bd} F$ and $\eta_j(\operatorname{bd} F)$ are both $(i-1)$-spheres. It follows that $\bigcup \operatorname{st} v_T$ is a closed i-ball, and a homeomorphism $\varphi_T : F \to \bigcup \operatorname{st} v_T$ that agrees with φ_j on $\operatorname{bd} F$ can be constructed. Now assume $G \neq \emptyset$. By lemma 4.2, $\operatorname{bd} F - \operatorname{int} G$ and therefore $\varphi_j(\operatorname{bd} F - \operatorname{int} G)$ are closed $(i-1)$-balls. Furthermore, $\operatorname{bd} G$ is an $(i-2)$-sphere. Let $\mathcal{L} \subseteq \mathcal{K}_i$ consist of the simplices $\sigma_{U \cup \{v_T\}}$, with $\sigma_U \subseteq \varphi_j(\operatorname{bd} G)$. $\bigcup \mathcal{L}$ is a closed $(i-1)$-ball, and a homeomorphism $\varphi'_T : G \to \bigcup \mathcal{L}$ that agrees with φ_j on $\operatorname{bd} G$ can be established. Now we are in the same situation as in the first case, and a homeomorphism $\varphi_T : F \to \bigcup \operatorname{st} v_T$ that agrees with φ'_T on G and with φ_j on $\operatorname{bd} F - \operatorname{int} G$ can be constructed. After adding all $F = \bigcap \mathsf{V}_{T,\mathbb{X}}$ with $\operatorname{card} T = m+1-i$ in this fashion, we get φ_i by combining all φ_T using the pasting lemma.

Observe that $\mathcal{K}$ and $\operatorname{sd} \mathcal{D}_{\mathbb{X}}$ contain a vertex for each $T \subseteq S$ with $\bigcap \mathsf{V}_{T,\mathbb{X}} \neq \emptyset$, so $\operatorname{vert} \operatorname{sd} \mathcal{D}_{\mathbb{X}} = \{\mathsf{b}_T \mid v_T \in \operatorname{vert} \mathcal{K}\}$. It follows that $f : \operatorname{vert} \mathcal{K} \to \operatorname{vert} \operatorname{sd} \mathcal{D}_{\mathbb{X}}$ defined by $f(v_T) = \mathsf{b}_T$ is a bijection. By construction of $\mathcal{K}$, if a collection of vertices belong to a common simplex in $\mathcal{K}$, then their images belong to a common simplex in $\operatorname{sd} \mathcal{D}_{\mathbb{X}}$. It follows that the simplicial map $g : \bigcup \mathcal{K} \to \bigcup \operatorname{sd} \mathcal{D}_{\mathbb{X}}$ implied by f is a homeomorphism. Therefore, $\bigcup \mathcal{K} \approx \bigcup \operatorname{sd} \mathcal{D}_{\mathbb{X}} = \bigcup \mathcal{D}_{\mathbb{X}}$, and the assertion of the theorem follows. □

REMARK. As illustrated in figure 4.1, the Delaunay simplicial complex restricted by $\mathbb{X}$ is related to the complex obtained from the chains of the regular CW complex defined by the Voronoi cells. Besides being smaller, an advantage of the Delaunay simplicial complex is that it naturally imbeds in the same space that contains $\mathbb{X}$ and S.

REMARK. For manifolds of dimension 3 or less, condition (B2) is implied by (B1) and the weaker requirement

(B2') $\bigcap \mathsf{V}_{T,\operatorname{bd}\mathbb{X}}$ is either empty or contractible.

Of course, this is interesting only for manifolds *with* boundary, for otherwise (B2) and (B2') are void. To see why (B2') is sufficient, let $1 \leq m \leq 3$ and suppose $\mathbb{X}$ and S satisfy the assumptions in lemma 4.1. Assume also that S satisfies condition (B1), that is, for every $\ell \leq m$ and every $T \subseteq S$ with $\operatorname{card} T = m+1-\ell$, $F = \bigcap \mathsf{V}_{T,\mathbb{X}}$ is either empty or a closed ℓ-ball. Note that

$$G = \bigcap \mathsf{V}_{T,\operatorname{bd}\mathbb{X}} = \operatorname{bd} F - \bigcup_{T \subset U \subseteq S} \operatorname{int} \bigcap \mathsf{V}_{U,\mathbb{X}}$$

is either empty or an $(\ell-1)$-manifold contained in an $(\ell-1)$-sphere, namely $\operatorname{bd} F$. For $\ell \leq 3$, the contractibility of $G \neq \emptyset$ implies that G is a closed $(\ell-1)$-ball; so (B2') implies (B2). For $m = 4$, (B2') implies (B2) if the Poincaré conjecture for 3-spheres is true, see e.g. [1].

5 General Topological Spaces

In this section, we generalize the result for compact manifolds to more general topological spaces. The requirements will automatically exclude spaces that cannot be expressed as the underlying space of a finite regular CW complex. In order to generalize theorem 4.3, we need extensions of the generic intersection and the closed ball properties. Let $\mathbb{X} \subseteq \mathbb{R}^d$ be a topological space and let $S \subseteq \mathbb{R}^d$ be a non-degenerate finite point set. S has the *extended closed ball property* for $\mathbb{X}$ if there is a regular CW complex $\mathcal{R}$, with $\mathbb{X} = \bigcup \mathcal{R}$, that satisfies the following properties for every $T \subseteq S$ with $\bigcap \mathsf{V}_{T,\mathbb{X}} \neq \emptyset$:

(E1) there is a regular CW complex $\mathcal{R}_T \subseteq \mathcal{R}$ so that $\bigcap \mathsf{V}_{T,\mathbb{X}} = \bigcup \mathcal{R}_T$,

(E2) the set $\mathcal{R}^\circ_T = \{\gamma \in \mathcal{R} \mid \operatorname{int} \gamma \subseteq \operatorname{int} \bigcap \mathsf{V}_T\}$ contains a unique cell, η_T, so that $\eta_T \subseteq \gamma$ for every $\gamma \in \mathcal{R}^\circ_T$,

(E3) if η_T is a j-cell then $\eta_T \cap \operatorname{bd} \bigcap \mathsf{V}_T$ is a $(j-1)$-sphere, and

(E4) for each integer k and each k-cell $\gamma \in \mathcal{R}^\circ_T - \{\eta_T\}$, $\gamma \cap \operatorname{bd} \bigcap \mathsf{V}_T$ is a closed $(k-1)$-ball.

We call η_T the *hub* of $\mathcal{R}^\circ_T$. Furthermore, S has the *extended generic intersection property* for $\mathbb{X}$ if for every $T \subseteq S$ and every $\gamma \in \mathcal{R}_T - \mathcal{R}^\circ_T$ there is a $\delta \in \mathcal{R}^\circ_T$ so that $\gamma \subseteq \delta$.

It is not difficult to see that if $\mathbb{X}$ is a compact manifold and S has the generic intersection and the closed ball properties for $\mathbb{X}$, then the extended generic intersection and the extended closed ball properties follow. Indeed, the regular CW complex, $\mathcal{R}$, required by condition (E1) consists of all non-empty sets $\bigcap \mathsf{V}_{T,\mathbb{X}}$ and $\bigcap \mathsf{V}_{T,\operatorname{bd}\mathbb{X}}$, $T \subseteq S$. Fix a subset $T \subseteq S$ and define $F = \bigcap \mathsf{V}_{T,\mathbb{X}}$ and $G = \bigcap \mathsf{V}_{T,\operatorname{bd}\mathbb{X}}$. If non-empty, F and G are closed balls, and if $G \neq \emptyset$ then $F \neq \emptyset$ and the dimension of

F exceeds the dimension of G by one. There are three possible cases. If $F = G = \emptyset$ then $\bigcap \mathsf{V}_{T,\mathbb{X}} = F = \emptyset$ and conditions (E2) through (E4) are void. If $F \neq \emptyset$ and $G = \emptyset$ then $\mathcal{R}_T^\circ = \{F\}$, and $\eta_T = F$ satisfies conditions (E2) and (E3); condition (E4) is void. If $F \neq \emptyset$ and $G \neq \emptyset$ then $\mathcal{R}_T^\circ = \{F, G\}$, $\eta_T = G$ satisfies conditions (E2) and (E3), and $\gamma = F$ satisfies condition (E4). In any case, the establishment of the homeomorphism in the proof of theorem 4.3 can be viewed as introducing a vertex v_T for η_T and connecting it to the simplices inductively constructed for the cells in $\mathcal{R}_T - \mathcal{R}_T^\circ$. This idea also works in the general case.

Let $\mathbb{X} \subseteq \mathbb{R}^d$ be a topological space, and let $S \subseteq \mathbb{R}^d$ be a non-degenerate finite point set that has the extended generic intersection and the extended closed ball properties for $\mathbb{X}$. To construct a homeomorphism between $\mathbb{X} = \bigcup \mathcal{R}$ and $\bigcup \mathcal{D}_{\mathbb{X}}$, we consider one subset $T \subseteq S$ at a time, in order of non-increasing cardinality. Inductively, we assume the cells in $\mathcal{R}_T - \mathcal{R}_T^\circ$ are already mapped homeomorphically to appropriate simplices. We extend the homeomorphism to the cells of $\mathcal{R}_T^\circ$, again inductively in order of non-decreasing dimension. We introduce a vertex, v_T, for the hub, η_T. If η_T is a k-ball, its boundary is a $(k-1)$-sphere, and by (E3) and the induction hypothesis, the cells in this $(k-1)$-sphere are already part of the homeomorphism. After connecting v_T to the simplices that correspond to the cells in bd η_T, we can extend the homeomorphism to η_T. Every other cell in $\mathcal{R}_T^\circ$ is treated the same way, only that the reason the homeomorphism can be extended is now a combination of the two induction hypotheses. This implies the generalization of theorem 4.3 to topological spaces other than manifolds.

THEOREM 5.1 Let $\mathbb{X} \subseteq \mathbb{R}^d$ be a topological space, and let S be a non-degenerate finite point set that has the extended generic intersection property for $\mathbb{X}$. If S has the extended closed ball property for $\mathbb{X}$ then $\bigcup \mathcal{D}_{\mathbb{X}} \approx \mathbb{X}$.

6 Remarks and Further Work

This paper is targeted at problems requiring the discretization of possibly complicated geometric objects in finite-dimensional Euclidean spaces. Such problems are abundant in the computational science literature, see e.g. Kaufmann and Smarr [12], where the discretization of continuous domains is common practice. The dimensions of the domain and the imbedding space can be the same, as e.g. common in fluid dynamics problems, or they can be widely different, as in the study of many dynamical systems. The restricted Delaunay simplicial complex introduced in this paper is a general method that produces simplicial discretizations in all cases.

The introduction of a general concept typically gives rise to many specific questions and directions for further research. We see three directions of progressively more basic work necessary to bring restricted Delaunay simplicial complexes closer to the targeted application areas. The first is the design of efficient algorithms that constructs the Delaunay simplicial complex of a finite point set, S, restricted by a domain or space, $\mathbb{X}$. Special cases under different assumptions on how $\mathbb{X}$ is specified are considered in [15, 4, 8]. The second direction is the design of methods that choose finitely many points resulting in good quality discretizations of $\mathbb{X}$. Such methods have a long history in the somewhat different context of finite element analysis, see e.g. [21]. The third direction is the study of maps from a problem specific domain to a space, possibly imbedded in higher dimensions, that depends on functionals studied over the domain or on approximate solutions thereof.

References

[1] R. H. BING. Some aspects of the topology of 3-manifolds related to the Poincaré conjecture. *Lectures in Modern Mathematics* 2 (1964), 93–128.

[2] R. BOTT AND L. W. TU. *Differentiable Forms in Algebraic Topology.* Springer-Verlag, New York, 1982.

[3] E. BRISSON. Representing geometric structures in d dimensions: topology and order. *Discrete Comput. Geom.* **9** (1993), 387–426.

[4] L. P. CHEW. Guaranteed-quality mesh generation for curved surfaces. *In* "Proc. 9th Ann. Sympos. Comput. Geom., 1993, 274–280.

[5] G. E. COOKE AND R. L. FINNEY. *Homology of cell complexes.* Princeton Univ. Press, 1967.

[6] B. DELAUNAY. Sur la sphère vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk* **7** (1934), 793–800.

[7] H. EDELSBRUNNER. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Heidelberg, Germany, 1987.

[8] H. EDELSBRUNNER. The union of balls and its dual shape. *In* "Proc. 9th Ann. Sympos. Comput. Geom., 1993", 218–231.

[9] H. EDELSBRUNNER AND E. P. MÜCKE. Simulation of Simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Comput. Graphics* **9** (1990), 66–104.

[10] V. GUILLEMIN AND A. POLLACK. *Differential Topology.* Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974.

[11] M. W. HIRSCH. *Differential Topology.* Springer-Verlag, New York, 1976.

[12] W. J. KAUFMANN III AND L. L. SMARR. *Supercomputing and the Transformation of Science.* Scientific American Library, New York, 1993.

[13] J. LERAY. Sur la forme des espaces topologiques et sur les points fixes des représentations. *J. Math. Pure Appl.* **24** (1945), 95–167.

[14] A. T. LUNDELL AND S. WEINGRAM. *The Topology of CW Complexes.* Van Nostrand, New York, 1969.

[15] T. MARTINETZ AND K. SCHULTEN. Competitive Hebbian rule forms manifold representing networks. *Neural Networks*, to appear.

[16] J. R. MUNKRES. *Topology: A First Course.* Prentice Hall, Englewood Cliffs, New Jersey, 1975.

[17] J. R. MUNKRES. *Elements of Algebraic Topology.* Addison-Wesley, Redwood City, California, 1984.

[18] F. P. PREPARATA AND M. I. SHAMOS. *Computational Geometry – an Introduction.* Springer-Verlag, New York, 1985.

[19] J. J. ROTMAN. *An Introduction to Algebraic Topology.* Springer-Verlag, New York, 1988.

[20] C. P. ROURKE AND B. J. SANDERSON. *Introduction to Piecewise-Linear Topology.* Springer-Verlag, Berlin, 1982.

[21] G. STRANG AND G. J. FIX. *An Analysis of the Finite Element Method.* Prentice Hall, Englewood Cliffs, New Jersey, 1973.

[22] G. F. VORONOI. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. *J. Reine Angew. Math.* **133** (1907), 97–178.

[23] W.-T. WU. On a theorem of Leray. *Chinese Math.* **2** (1962), 398–410.

[24] C. K. YAP. Symbolic treatment of geometric degeneracies. *J. Symbolic Comput.* **10** (1990), 349–370.

Almost Optimal Set Covers in Finite VC-Dimension
(Preliminary Version)

HERVÉ BRÖNNIMANN*
Department of Computer Science
Princeton University
Princeton, NJ 08544, USA
E-mail: hbr@cs.princeton.edu

MICHAEL T. GOODRICH†
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218, USA
E-mail: goodrich@cs.jhu.edu

Abstract

We give a deterministic polynomial time method for finding a set cover in a set system $(X, \mathcal{R})$ of VC-dimension d such that the size of our cover is at most a factor of $O(d \log(dc))$ from the optimal size, c. For constant VC-dimension set systems, which are common in computational geometry, our method gives an $O(\log c)$ approximation factor. This improves the previous $\Theta(\log |X|)$ bound of the greedy method and beats recent complexity-theoretic lower bounds for set covers (which don't make any assumptions about VC-dimension). We give several applications of our method to computational geometry, and we show that in some cases, such as those that arise in 3-d polytope approximation and 2-d disc covering, we can quickly find $O(c)$-sized covers.

1 Introduction

A *set system* $(X, \mathcal{R})$ is a set X along with a collection $\mathcal{R}$ of subsets of X, which are sometimes called *ranges* [24]. Such entities have also been called *hypergraphs* and *range spaces* in the computational geometry literature (e.g., see [5, 9, 10, 11, 12, 13, 14, 15, 18, 23, 24, 33, 35, 34, 40, 36, 38, 39]), and they can be used to model a number of interesting computational geometry problems.

There are a host of *NP*-hard problems defined on set systems, with one of the chief such problems being that of finding a *set cover* of minimum size (e.g., see [20, 22]), where a set cover is a subcollection $C \subseteq \mathcal{R}$ whose union is X and the size of C is simply the number of sets in C. A related (in fact, equivalent) problem is that of finding a *hitting set* of smallest size, where a hitting set is a subset $H \subseteq X$ such that H has a non-empty intersection with every set R in $\mathcal{R}$. There are a number of problems that can be reduced to these two problems, and many of these problems are formulated as computational geometry problems. Thus, our interest is in finding minimum set covers and smallest hitting sets as quickly as possible.

Unfortunately, the corresponding decision problems, SET COVER and HITTING SET, as we will call them, are both *NP*-complete [22, 28]. Moreover, even if each element of X is contained in just two sets, the HITTING SET problem is still *NP*-complete [28] (and is known as VERTEX-COVER). Thus, unless $P = NP$, if we desire a polynomial-time algorithm, we must content ourselves with an approximation algorithm, which, if we let c denote the size of a minimum set cover, produces a set cover of size αc for some (hopefully) small approximation factor α. The only known polynomial-time approximations algorithms for SET COVER with guaranteed performance ratios are the greedy algorithm [16, 27, 31], which achieves an approximation factor of $\alpha = (1 + \ln A)$, where A denotes the size of the largest set in R, and an algorithm of Hochbaum [25], which achieves an approximation

*Supported in part by NSF Grant CCR-90-02352 and Ecole Normale Supérieure.

†This research supported by the NSF and DARPA under Grant CCR-8908092, and by the NSF under Grants IRI-9116843 and CCR-9300079.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

factor of $\alpha = \max_{x \in X}\{|\mathcal{R}_x|\}$, where $\mathcal{R}_x$ denotes the set of all $R \in \mathcal{R}$ containing x. Note that in the former case α can be as large as $(1+\ln|X|)$ and in the latter α can be as large as $|\mathcal{R}|$. Thus, as worst case bounds, the greedy algorithm achieves the best approximation factor, and Johnson [27] shows there are set systems where the greedy algorithm does indeed produce a set cover with approximation factor $\alpha = \Omega(\ln|X|+1)$. Interestingly, Lund and Yannakakis [32] have recently shown that, unless $NP \subseteq DTIME[n^{\text{poly}\log n}]$, no polynomial time algorithm can approximate Set Cover within a factor better than $\alpha = \frac{1}{4}\ln|X|$ in the worst case, and, unless $NP \subseteq DTIME[n^{\log\log n}]$, Bellare *et al.* [4] showed that no polynomial time algorithm can approximate Set Cover within a factor better than $\alpha = \delta \ln|X|$ in the worst case, for any constant $\delta < 1/8$.

Our results. In this paper we give a (deterministic) polynomial-time algorithm that finds a set cover of size within a factor of $\alpha = O(d\log(dc))$ of the size c of the minimum set cover, where d stands for the VC-exponent of the set system (see Section 2 for definitions). Thus, for set systems with bounded VC-exponent, we actually beat the complexity-theoretic lower bounds [4, 32] (which make no assumptions about the VC-exponent). In section 4, we indicate some examples on which Hochbaum's method [25] or the greedy methods [16, 27, 31] perform poorly, and show that our method outperforms them in some cases.

Our algorithm, which is actually for the dual Hitting Set problem, is based upon a deterministic analogue of a randomized *natural selection* technique used by Clarkson [17, 19], Littlestone [30], and Welzl [44]. This technique can be viewed as "algorithmic Darwinism," for we iteratively select, in polynomial-time, a small-sized subset of X (known in the literature as an ε-net [24]) that intersects all highly-weighted sets in $\mathcal{R}$, and, if this set is not a hitting set, then we increase the weight of the elements in an $R \in \mathcal{R}$ missed by this set. By continuing this process over several "generations" we guarantee that the "fittest" elements, which belong to the optimal hitting set, eventually are included. Fortunately, the number of generational iterations we must perform is at most $O(c\log(|X|/c))$; hence, this approach yields a (relatively simple) polynomial-time algorithm.

We show the utility of our approximation algorithm by giving several applications of our method to problems in computational geometry and learning theory, including polytope approximation, geometric point-probe decision tree construction, and disk cover. Our methods improve the running time and/or the approximation factors of previous methods. In fact, for 3-d polytope approximation and 2-d disc covering we show how to adapt our method to achieve a constant approximation factor. We also describe an implementation of our approach, and show that it indeed beats the greedy method in practice in some cases.

Before we describe our methods, however, let us first review the relevant properties and definitions regarding the VC-dimension notion.

2 Set Systems of Finite VC-Dimension

In this section we recall the basic facts about set systems of finite VC-dimension. Let $(X, \mathcal{R})$ be a given set system. Given $Y \subseteq X$, all the subsets of Y obtained as the intersection of Y and R ranging over $\mathcal{R}$, form a set system called the system induced by $\mathcal{R}$ on Y, and denoted by $\mathcal{R}_{|Y}$. Y is *shattered* by $\mathcal{R}$ if $\mathcal{R}_{|Y} = 2^Y$. $(X, \mathcal{R})$ is said to have *VC-dimension d*, if d is the smallest integer such that no $d+1$ point subset $Y \subseteq X$ can be shattered. If Y is finite, it is well known [42, 43] that the number of sets of $\mathcal{R}_{|Y}$ is less than $\binom{|Y|}{0}+\cdots+\binom{|Y|}{d} \leq |Y|^d$, where d is the VC-dimension of $(X, \mathcal{R})$. We call the *VC-exponent*[1] the infimum of all numbers s such that $|\mathcal{R}_{|Y}| = O(|Y|^s)$ for any finite subset Y of X. The aforementioned result shows that $s \leq d$, but equality does not always occur, as there are systems in which the VC-exponent is non integral [2]. However, the VC-dimension is finite if and only if the VC-exponent is finite as well; s never takes values in the interval $[0, 1]$, and it is 0 if an only if $\mathcal{R}$ is finite [2] (whereas d is 0 if and only if $\mathcal{R}$ consists of one set). In particular, the VC-exponent concept only makes sense for an infinite set system, whereas the VC-dimension is more general. Consider, for the sake of an example, a common set system arising often in computational geometry applications—the *halfspace* set system. In this

[1] This value has also gone by the names *real density* [2] and *scaffold dimension* [23].

system X is taken as a set of halfspaces in $\mathbb{R}^d$ and $\mathcal{R}$ is taken to be all combinatorially distinct ways if intersecting halfspaces in X by a simplex. It is well known (e.g., [2]) that this system has a VC-dimension and a VC-exponent of d.

The dual set system $(\mathcal{R}, X^*)$ is defined by $X^* = \{\mathcal{R}_x : x \in X\}$, where $\mathcal{R}_x$ consists of all the sets $R \in \mathcal{R}$ that contain x. One way to look at dual systems is to consider $(X, \mathcal{R})$ as a (possibly infinite) boolean matrix M that has a column for each $x \in X$ and each row corresponds to an incidence relation for a set $R \in \mathcal{R}$, and view the dual as a transpose of this matrix. It is also well known (e.g., see Assouad [2]) that the VC-dimension d^* of the dual $(\mathcal{R}, X^*)$ is less than 2^{d+1}, where d is the VC-dimension of the primal $(X, \mathcal{R})$.

Let $(X, \mathcal{R})$ be a set system. If a subset $N \subseteq X$ intersects each set $R \in \mathcal{R}$ of size bigger than $\varepsilon|X|$, then we call N an ε-net [24]. As has been observed by Haussler and Welzl [24], set systems of VC-dimension d admit $(1/r)$-nets of size $O(dr\log(dr))$. This bound has been improved by Blumer *et al.* [7] to $O(dr\log r)$ and this has been proved tight by Komlós *et al.* [29]. We can generalize this definition by putting an additive[2] weight function w on 2^X. In this formulation, an ε-net is required to intersect every set of weight at least $\varepsilon w(X)$. To allow for efficient computation, we need that the set system be described somewhat more efficiently than by the list of its subsets.

Definition 2.1 *We say that a set system $(X, \mathcal{R})$ has a* subsystem oracle of degree D *if there is an algorithm which, given a subset $Y \subseteq X$, returns $(Y, \mathcal{R}_{|Y})$ (as a boolean matrix) in time $O(|Y|^{D+1})$. It is a* witnessed *oracle if, for any set R of $\mathcal{R}_{|Y}$, it can provide a set R' of $\mathcal{R}$ such that $R = R' \cap Y$ in $O(|X|)$ time.*

If $(X, \mathcal{R})$ has a subsystem oracle of degree D, and VC-exponent d, it is clear that $d \leq D$. Under this assumption, it has also been shown [9] that one can find a $(1/r)$-net for $(X, \mathcal{R})$ of size $O(dr\log(dr))$ in time $O(d)^{3D}r^D\log^D(rd)|X|$, for both the uniform and weighted cases.

3 The Main Algorithm

The goal of this section is to prove our main theorem. Let s be a non decreasing function, let $n = |X|$, and let $m = |\mathcal{R}|$.

Definition 3.1 *An* s-net finder *for $(X, \mathcal{R})$ is an algorithm A that, given r and a weight distribution w on X, returns a $(1/r)$-net of size $s(r)$ for $(X, \mathcal{R})$ with weights w. Also, a* verifier *is an algorithm B that, given a subset $H \subseteq X$, either states (correctly) that H is a hitting set, or returns a nonempty $R \in \mathcal{R}$ such that $R \cap H = \emptyset$.*

Using size(x) to denote the length of the encoding of an object x, let us say that A runs in $T_A(\text{size}(X, \mathcal{R}), \text{size}(w), r)$ time[3] and that B runs in $T_B(\text{size}(X, \mathcal{R}), \text{size}(H))$ time. We will suppose that both T_A and T_B are (non constant) polynomials, so that $T(O(x)) = O(T(x))$ and $\sum_{k \leq x} T(2^k) = O(T(2^x))$.

Theorem 3.2 *Let $(X, \mathcal{R})$ be a set system that admits both an s-net finder A and a verifier B. Then there is an algorithm $\mathcal{A}(A, B)$ that computes a hitting set of size at most $s(4c)$, where c stands for the size of an optimal hitting set. The time taken by $\mathcal{A}$ is $O(c\log(n/c)\,(T_A(nm, n\log(n/c), c) + T_B(nm, c))$.*

If X has finite VC-dimension, then we may implement the net finder using the greedy method [13], and the verifier by inspecting all of $(X, \mathcal{R})$, both in polynomial time (in either the real RAM or bit models). But under standard computational assumptions, there are better implementations of the net finder and verifier.

Corollary 3.3 *Let $(X, \mathcal{R})$ be a set system given by a witnessed subsystem oracle of degree D, which admits a hitting set of size c. Let d stand for the VC-exponent of $(X, \mathcal{R})$, and assume $\emptyset \notin \mathcal{R}$. Then one can find a hitting*

[2] The term additive refers to the fact that $w(Y) = \sum_{y \in Y} w(y)$, with the usual abuse of notation $w(y) = w(\{y\})$.

[3] Throughout the literature in computational geometry, the unit-cost (or real RAM) model is usually assumed. In this case, size(w) is simply n. However, due to to the particular relevance of our algorithm in the realm of NP-hardness, and since the weights can (and will) become exponential in the forthcoming algorithm, we have to be careful that the algorithm still runs in polynomial time in the bit model. As it turns out, our running time is the same in both models, but the choice of the encoding and the value of T_A are different.

set for $(X, \mathcal{R})$ of size $O(dc \log(dc))$ in $T(|X|) = O\left(c^{2D+1} \log^D(dc) \log(|X|/c)|X|\right)$ time in the real RAM model, or $O\left(cT(|X|) \log(|X|/c)\right)$ time in the bit model.

Proof: It suffices to show how to implement the net finder and the verifier and show their complexity bound. But one can find a $(1/r)$-net for $(X, \mathcal{R})$ of size $O(dr \log(dr))$ in $O(d)^{3D} r^{2D} \log^D(rd)|X|$ time, for both the uniform and weighted cases, using the algorithm of Brönnimann, Chazelle, and Matoušek [9], in the real RAM model. In the bit model, the complexity of the weights would have to be taken into account, which may add at most an $O(c \log(|X|/c))$ factor to the running time, as proven in the next section. As for the verifier, one simply runs the witnessed subsystem oracle for H. If the oracle fails to list $\emptyset$ as being in $\mathcal{R}_{|H}$, we may conclude that H is a hitting set. Otherwise, we ask for a witness R of the fact that $\emptyset \in \mathcal{R}_{|H}$. Thus $R \cap H = \emptyset$, and R is nonempty as $\emptyset \notin \mathcal{R}$. The time spent by the verifier is $O(|H|^{D+1} + |X|)$. Plugging these bounds into Theorem 3.2 yields the corollary. □

Remarks. (1) If we change the definition of the VC-exponent to fit better the VC-dimension concept, by requiring that for all finite $Y \subseteq X$ we have $|R_{|Y}| = O\left(\binom{|Y|}{0} + \cdots + \binom{|Y|}{d}\right)$ which is also $O(|Y|/d + 1)^d$, then we can reduce the size of the hitting set to $O(dc \log c)$ [14].
(2) For a particular c, the algorithm will either say that there is no hitting set of size c, or it will output a hitting set of size $O(dc \log(dc))$, which, of course, is different than saying that there *is* a hitting set of size c.
(3) The corollary can be stated with d as the *dual* VC-exponent, with an oracle for the dual set system, and as yielding a *set cover* of size $O(dc \log(dc))$.

We now turn to the proof of Theorem 3.2.

3.1 The algorithm

Let us assume, for the time being, that we know the size c of a smallest hitting set (we will show later why this is a reasonable assumption).

Our strategy can be thought of in terms of evolutionary biology, in that it is based upon a notion of "survival of the fittest." Intuitively, we want to simulate the growth of a population where some elements are advantaged because they hit more sets than others. The idea is to put weights on the elements (initially uniformly) and use the net finder A to select a $(1/2c)$-net of size $s(2c)$. If it doesn't hit a particular set $R \in \mathcal{R}$, as returned by the verifier B on the net, we double the weights of the points in R. Because of the definition of a hitting set, at least one point in an optimal hitting set falls in R, and has its weight doubled. However, the property of a $(1/2c)$-net implies that the weight of R is at most a fraction $1/2c$ of the total weight, and the total weight does not increase too much. Therefore, we soon expect an optimal hitting set to be included in the chosen set. The next lemma shows that this is indeed the case.

Lemma 3.4 *If there is a hitting set of size c, the doubling process cannot iterate more than $4c \log(n/c)$ times, and the total weight will not exceed n^4/c^3.*

Proof: Our proof follows arguments of Clarkson [17, 19], Littlestone [30], and Welzl [44]. Let H be a hitting set of size c. Because the set R returned by B at each iteration satisfies $w(R) \leq w(X)/2c$, the weight of X is not multiplied by more than a factor $1 + 1/2c$ in any iteration. However, $H \cap R$ is not empty, by definition of H. Therefore, after k iterations, if each $h \in H$ has been doubled z_h times, we have

$$\begin{array}{lclcl} w(X) & \leq & n\,(1+\dfrac{1}{2c})^k & \leq & n\,e^{\frac{k}{2c}}, \qquad \text{and} \\ w(H) & = & \displaystyle\sum_{h \in H} 2^{z_h}, & & \text{where } \displaystyle\sum_{h \in H} z_h \geq k. \end{array}$$

Using the convexity of the exponential function, we conclude that $w(H) \geq c2^{k/c}$. Since $w(H) \leq w(X)$, we finally have

$$c 2^{\frac{k}{c}} \leq n\,e^{\frac{k}{2c}} \leq n\,2^{\frac{3k}{4c}},$$

from which $k \leq 4c \log(\frac{n}{c})$ follows. The bound of n^4/c^3 on $w(X)$ is an immediate consequence. □

If the process exceeds this guaranteed number of iterations, that only means that there is no cover of size c. Thus, we can use this procedure to actually determine an approximate bound for c. To start with, we conjecture a value c' of c, which initially can be set to 1. In general, if our routine fails to find a hitting set, there is no hitting set of size c', so we increase c' by a factor of two. At the stage when we find a hitting set, $c' \leq 2c$, so that the hitting set returned is of size at most $s(4c)$.

3.2 ε-nets with or without weights?

Typically, ε-net algorithms are designed for the uniform case not for the weighted case. Here we mention a simple method for reducing the weighted case to an unweighted one, as outlined by Matoušek [35]. First scale the weights such that $w(X) = n$. Then take $\lfloor w(x) + 1 \rfloor$ copies of each element $x \in X$. Note that the multiset X' thus obtained contains all the elements of X and has a cardinal of at most $2n$. Take then an ε-net for the set X': it is also an ε-net for the original set X with weights w.

3.3 Time analysis

For a given c', it is clear that there can be at most $4c' \log(n/c')$ iterations, each of which takes time $O\left(T_A(\text{size}(X,\mathcal{R}), \text{size}(w), c') + T_B(\text{size}(X,\mathcal{R}), c')\right)$. But the total weight can never exceed n^4/c^3, which proves that $\text{size}(w) = O(n \log(n/c'))$. When c' increases geometrically, by our choice of polynomial functions for T_A and T_B, the sum of all running times is dominated by their last term, for which $c \leq c' \leq 2c$. Finally, a standard encoding of $(X, \mathcal{R})$ gives that $\text{size}(X, \mathcal{R}) = O(nm)$, concluding the proof of Theorem 3.2.

Note that in real RAM model, the size of the weights is irrelevant, and one simply accounts for the number of arithmetic operations and not their complexity. As argued in Section 6, the weights almost never go past a few digits in practice, so such an assumption is not unrealistic. In any case, since our weights are always multiples of 2, we can encode each weight using a register of $O(\log n)$ bits and we can use such registers to perform any weight arithmetic in $O(c \log(n/c))$ time (by Lemma 3.4).

4 Lower bounds for other methods

In this section we give some specific cases where our method improves the previous methods. We begin with the greedy method.

4.1 The greedy method

The following example has been communicated to us by Jiří Matoušek (private communication). Let $S = \{p_{i,j} : i = 1, 2, j = 1 \ldots 2^{p+1} - 1\}$ be a set of $n = 2^{p+2} - 2$ distinct points, and let its valid subsets be $S_j = \{p_{i,k} : i = 1, 2, k = 2^j \ldots 2^{j+1} - 1\}$, for $i \in \{1, 2\}$, $j \in \{0 \ldots p\}$, as well as the two particular sets $T_i = \{p_{i,j} : j = 1 \ldots 2^{p+1} - 1\}$ (see Figure 1). The greedy method will pick the cover $(S_j)_{j=1\ldots p}$, which is of size $p = \log(n+2) - 2$. The optimal cover is $(T_i)_{i=1,2}$ and is of size 2. The dual VC-dimension of our example is 2, as every point is covered by exactly 2 subsets, and our algorithm (in a dual setting to obtain a set cover) will pick the optimal cover for a value of $c = 2$.

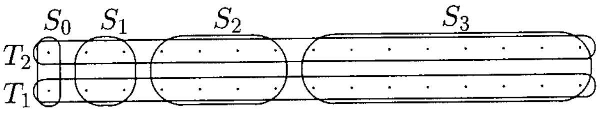


Figure 1: A greedy method's worst case that has finite VC-dimension.

4.2 Hochbaum's method

In Hochbaum's method, one denotes the set system of m sets over n elements as a 0–1 (n, m)-matrix A each column A_j of which is the incidence vector of one of the sets. The algorithm then solves the linear program $\max e_n^T.y$ subject to $y^T.A \leq e_m, y \geq (0, \ldots, 0)$, where e_k is a k-vector whose components are all 1. Let J denote the sets of tight constraints at the optimum y^* (in mathematical notation, $j \in J$ if and only if $(y^*)^T.A_j = 1$). Then J is a set cover within f times the optimum, where f is the maximum row sum of A.

Take k to be any constant. Let us put $n = \binom{m}{k}$ and take for A the (n, m)-matrix whose rows are made up of all possible vectors with $m-k$ 1's and k 0's (in no particular order). The corresponding set system has n elements and m sets. By symmetry, the method will pick all the sets in the cover, which gives a cover of size m, guaranteed within $m-k$ off the optimum. Any $k+1$ elements will constitute an optimal cover.

This example has both dual VC-exponent and dual VC-dimension k, so our algorithm will return a cover of size $O(k^2 \log k)$.

Remark. Interestingly enough, the bad example for the greedy (resp. Hochbaum's) method can be solved efficiently with our as well as Hochbaum's (resp. the greedy) method. We don't know of an example for which our method would outperform both of them.

5 Applications

In this section, we cover mainly computational geometry applications, therefore it is customary to assume the real RAM model where all the integers and their elementary arithmetic operations have unit complexity.

5.1 Separating polyhedra

Let $Q \subseteq P$ be two convex polytopes in $\mathbb{R}^d$. The problem of finding a separator polytope between P and Q with as few facets as possible is believed to be *NP*-hard, even in the three-dimensional case [21], but one can find good approximations for it. In [41] Mitchell and Suri cast the problem as a hitting set problem. Let $\mathcal{H} = \mathcal{H}(Q)$ denote the set of supporting hyperplanes of Q. Let the *cover set system* be $(\mathcal{H}, \{\mathcal{H}_p : p \in \partial P\})$, where, for any point p, $\mathcal{H}_p$ consists of those hyperplanes in $\mathcal{H}$ for which p and Q lie on opposite sides. (For definiteness, we agree that if p is on h, then h is not in $\mathcal{H}_p$.) They show that a hitting set for the cover set system gives a subset of facets of Q whose bounding halfspaces define a polytope between Q and P. A polytope Q' between Q and P such that each facet of Q' contains a facet of Q is called *canonical*. The smallest (with respect to the number of facets) canonical polytope contained in P can be obtained from a minimal hitting set of this set system and is within d times the optimal separating polyhedron (in number of faces). Therefore the greedy strategy returns a polyhedron within $O(d^2 \log |Q|)$ of the optimal. In a recent twist, Clarkson [19] applies ideas he used for small-dimensional linear programming to give a polynomial-time randomized method that produces an approximation within $O(d^2 \log(dc))$ of the optimal c. In fact, the algorithm of this paper is a deterministic analogue of Clarkson's method in our more general framework[4]. Interestingly, however, for the important case of $d = 3$, we are able to improve on Clarkson's result to achieve an approximation factor that is $O(1)$.

But let us first describe our method for general $d \geq 4$. We need to exhibit an appropriate net finder and a verifier. Note that since P contains Q, $\emptyset$ is not in the cover set system. The basic observation is that the cover set system is a subsystem of the halfspace set system, and therefore has VC-dimension and VC-exponent d (since $d+1$ hyperplanes define at most $2^{d+1}-1$ regions of the space, and the complexity of an arrangement of m halfspaces is $O(m^d)$). Therefore our hope is to find an s-net finder, where $s(x) = O(dx \log(dx))$.

To find a $(1/r)$-net for the weighted cover set system, we simply use the algorithm of Brönnimann *et al.* [9], which computes a weighted $(1/r)$-net in $O(nr^d \log^d(dr))$ time, using the reduction of Section 3.2 to take care of the weighted case. This provides a $(1/r)$-net for the more general set system consisting of *all* halfspaces, but this is *a fortiori* a $(1/r)$-net for the cover set system, and the bound on its size is still $O(dr \log(dr))$.

As for the verifier, let $Y^\cap$ denote the canonical polytope with k faces associated with the hyperplanes of some $Y \subseteq \mathcal{H}$ of size k. In particular, $\mathcal{H}(Q)^\cap = Q$. All we are asking for is whether $Y^\cap$ is entirely contained in P, or else to give a point of ∂P which is in $Y^\cap$. We can simply compute the intersection of all the halfspaces defined by the planes $Y \cup \mathcal{H}(P)$ and test whether there is a facet that belongs to P. If so, we can return $\mathcal{H}_p$ where p is, e.g., the centroid of this face. Otherwise, $Y^\cap$ is certainly contained in P.

We can find a $(1/r)$-net in $O\left(r^d \log^d(dr)|Q|\right)$ time and verify k halfspaces in $O(k + |P|)^{\lfloor \frac{d}{2} \rfloor}$ time ($d \geq 4$) or in $O((k + |P|) \log(k + |P|))$ time ($d = 2, 3$). Plugging those bounds into our main theorem, we get that

Theorem 5.1 *Let $Q \subseteq P$ be two convex nested polytopes in $\mathbb{R}^d$ ($d \geq 4$) with a total of n facets.*

[4] Clarkson's exposition is restricted to polytopal problems.

It is possible to find a separator of size within $O(d^2 logc)$ of the optimum c, in a deterministic time of $O\left(n^{\lfloor \frac{d}{2} \rfloor} + c^d n \log^d(dc)\right) c \log(n/c)$, which is always $O\left(n^{d+2} \log^d n\right)$.

Note that the case where $c = \Omega(n^\varepsilon)$ is of little interest, because, in this instance, our algorithm offers no better a performance ratio than the greedy method (at least asymptotically). Thus we could make the algorithm faster (for the same guaranteed ratio) by switching to the greedy method when c becomes too large.

But in three dimensions, we can actually do much better. In particular, we recall from [38, 37] that the three dimensional halfspace set system admits ε-nets of size $O(1/\varepsilon)$, and as indicated in Section 3.2, their algorithm can be extended to the weighted case as well. The cost of the computation of a $(1/c)$-net of size $O(c)$ is $O(nc)$, if $c \leq n^\alpha$ [37], and the cost of verifying is $O(n \log n)$ as argued above. Therefore we obtain the following strengthening of our previous theorem:

Theorem 5.2 *Let $Q \subseteq P$ be two nested polyhedra in $\mathbb{R}^3$ with a total of n facets, one of them being convex. It is possible to deterministically find a separator of size within $O(1)$ from the size c of an optimum separator in $O(nc(c + \log n) \log(n/c))$ time, if $c \leq n^\alpha$ for some $\alpha > 0$.*

In particular, it should be noted that the running time is always $O(n^{1+2\alpha} \log n)$, which improves on the $O(n^4)$ time bound of Mitchell and Suri's method [41] for the general case. However, for large $c \geq n^\alpha$, we have to switch to a $O(n^3)$ method to find the net [38], yielding $O(cn^3 \log(n/c))$ time. Note that this is still $O(n^4)$, whereas the time consumed by the greedy method is strictly $O(n^4)$ (though Mitchell and Suri reduce it to $O(n^3)$ when both polyhedra are convex.)

5.2 Decision Trees

In [1], Arkin *et al.* describe how to construct a point-probe decision tree for a set of non-degenerate polygons in the plane whose height is $s - 1 + \lceil \log(k/(s-1)) \rceil$, when given a hitting set of size s. For some applications, the result serves to decide the position of a polygon in a scene [1]. While their approach is more general, the next result shows that we can get better performance ratio in the height of a decision tree for non-degenerate (i.e., general position) inputs. We define the *point set system* of a family S of plane objects by all subsets S_p of S for all points p, where S_p consists of all objects in S containing p.

Lemma 5.3 *Let P be a polygon in the plane, S be a family of congruent copies of P, and let $(S, \mathcal{R})$ be the point set system of S. Then $(S, \mathcal{R})$ has VC-exponent at most 2 (the constant 2 cannot be improved in general), and admits a witnessed subsystem oracle of degree 2. The exponent is still 2 if P convex.*

Proof: Clearly, the number of sets of $(S, \mathcal{R})$ is at most the number of cells is the arrangement of S. If all m polygons in S have k vertices each, their arrangement has complexity $O(k^2m^2)$, which corresponds to the complexity of the arrangement of their supporting lines. The complexity drops to $O(km^2)$ if the polygons are convex, which provides somewhat better constants for the shatter function. However, it is easy to come up with examples for which those bounds are tight and also reflect the number of different subsets in the point set cover. This also provides the construction of the witnessed oracle: The oracle constructs the arrangement and simply picks a point in each of the cells. If asked to provide a witness, the point is tested against *all* the polygons, in $O(m \log k)$ time. (Here, we treat k as a constant, and we are only interested the number of elements of S.) □

Remark. Clearly, the VC-dimension depends on the number of sides of P. If S consists of congruent copies of a single convex set, even a convex polygon with infinitely many sides, then its point set system $(S, \mathcal{R})$ need not have finite VC-exponent (resp. VC-dimension).

Our algorithm, combined with the above observation, and the result of Arkin *et al.* [1] gives us a decision tree of smaller asymptotic depth than the greedy method for the non-degenerate arrangement of Arkin *et al.*[5] Moreover, our method can also be used to derive point-probe decision trees

[5] For degenerate arrangements they design a new greedy strategy, which seems not to be describable as a set cover strategy.

of improved depth for any set of general-position k-gons in the plane, for constant k, since this point set system also has bounded VC-dimension (as the proof above shows as well).

5.3 Disc cover

Let S be a set of n points in the plane and let $\mathcal{D}$ be a family of discs. The *disc cover* problem is to find a minimum number of discs in $\mathcal{D}$ that cover the points in S [26]. This problem is motivated by VLSI design criteria as well as viewing problems in astronomy. Matoušek *et al.* [38] show that this set system admits $O(r)$ sized $(1/r)$-net and, as indicated in Section 3.2, one can extend their algorithm to weighted point sets. The resulting algorithm runs in $O(rn \log n)$ time, and implies the following:

Theorem 5.4 *Let S be a set of points in the plane, and $\mathcal{D}$ be a set of discs whose union contains S. It is possible to find a disc cover of S from $\mathcal{D}$ (a subset of $\mathcal{D}$ whose union still contains P) whose size is no more than $O(1)$ times the optimal size c of such a cover in $O(c^2 n \log n \log(n/c))$ time.*

Note that our method never takes more than $O(n^3 \log n)$ time. This bound contrasts the bound of Hochbaum and Maas [26] for finding a constant-factor approximation to a disc cover, as their method requires all the discs in $\mathcal{D}$ to have the same radius and, even when all the parameters in their method are optimized for the running time, their method takes $O(n^5)$ time.

5.4 Learning a Union of Halfspaces in Fixed Dimension

Let H be a set of s halfspaces in $\mathbb{R}^d$, and write H^+ for their union, and H^- for its complement. Let D be an (unknown) probabilistic distribution on $\mathbb{R}^d$. A *learning algorithm A* for H^+, given m examples drawn from D, outputs a *hypothesis G* such that, with confidence $1 - \delta$ (on the examples fed to the algorithm), G verifies $D(H^+ \Delta G^+) < \varepsilon$. ($\Delta$ denotes the symmetric difference.) An example is labeled positive if it falls in H^+, negative otherwise. This ensures that, for subsequent examples, the label as computed with G will be correct (according to H) with probability at least $1 - \varepsilon$.

It has been proved by Blum and Rivest [6] that this problem admits no proper learning algorithm (for which the hypothesis also consists of s halfspaces) whose running time is polynomial in d, s, ε^{-1}, δ^{-1}, even if $s = 2$ (unless $P = NP$). However, Baum [3] argues that this only shows that we should look for hypotheses with more halfspaces. Indeed, Blumer *et al.* [7] and Baum [3] have proposed algorithms that output a hypothesis of size $O(s \log m)$. These algorithms are polynomial in fixed dimension (however, the dependence in the dimension is exponential). Unfortunately, with the purpose of training a neural net in mind, we see that each halfspace of the hypothesis corresponds to a gate of the neural net, and therefore the size of the neural net increases with precision.

The algorithm of Blumer *et al.* proceeds by first labeling the examples, then forming a set system of the halfspaces containing only positive examples, and finally returning a small set of halfspaces covering all the positive examples. For this last step we can use our method: The hypothesis is guaranteed to have size $O(ds \log(ds))$, independent of the number of examples. Moreover, we can improve on the results in two and three dimensions. This proves

Theorem 5.5 *For any fixed $d > 1$, given s halfspaces H in $\mathbb{R}^d$, and any distribution on points in $\mathbb{R}^d$, there exists a neural net of size $O(ds \log(ds))$ which can be trained, in time polynomial in s, ε, δ and with confidence $1 - \delta$, to recognize whether a point belongs to the union of H with a probability of error less than ε. The size of the neural net can be brought down to $O(s)$ if $d = 2, 3$.*

The result, of course, is also valid for learning the union of concepts taken from a concept class that has finite VC-dimension.

6 Experimental Results

Our algorithm has been implemented and the results will be described in [8]. There are many possible choices for implementation, as we have many ways to compute an ε-net in practice, and they result in different performances.

The first consists in choosing a random sample of size $O(dc \log c)$. It turns out that this performs badly against the greedy method, as the values of n

we consider are never big enough to justify the use of our method with the random sampling against the greedy method.

The second possible choice is to use a computed ε-net, with the algorithm of Chazelle and Matoušek [14], and this seems to yield results comparable to the greedy method.

The third method of choice is to compute the net using the greedy method [13]. Even though this does not guarantee the same performance as the second choice, it performs well in practice, and never returns anything bigger than that returned by the greedy set cover method. Moreover, the lower bound as returned by our method is substantially higher than that returned by the greedy method. Therefore, if it doesn't produce better hitting sets, at least it is able to give better lower bounds in practice.

For our three choices, the fear that weights might increase exponentially and therefore require a multi-precision treatment is completely dispelled by empirical observation. In the largest example we tried (of size about 1000^2), after some 700 iterations, the maximum weight was a bare 16384—as opposed to a possible 2^{700}. This strongly suggests that the normal precision (of 32 bits) will suffice in all reasonable instances of the problem. This is due, of course, to the fact that only small weights tend to be doubled through the algorithm.

One last issue concerns the generation of "random" set systems of finite VC-dimension. As this is a hard problem, we simply generated random set systems in such a way that the generation process itself guaranteed the finite (parameterizable) VC-dimension. We also tested the method on more natural set systems such as the cover set systems of two polyhedra.

7 Conclusions

We have shown an approximation algorithm for the hitting set and set cover problems, and proved that it performs well if the VC-dimension of the set system is bounded. Furthermore, our algorithm "defeats" known complexity-theoretic lower bounds [4, 32] (which make no assumptions about the VC-exponent), by being adaptive: the competitive ratio depends on the size of the optimal solution. We have exhibited a variety of computational geometry problems that reduce to or use hitting sets, and for which the set systems have bounded VC-dimension.

Our algorithm has been implemented, and the practical results are encouraging.

The main open question pertaining to this research is whether our algorithm can be made to run in parallel in polylogarithmic time and polynomial number of processors. It seems that the main problem lies in the iteration process: if we are to double the weights of many sets at once, we are unable to enforce that the weights of many points in an optimal hitting set double at each stage.

An ε-net is a hitting set for the heavy subsets. The algorithm can be thus adapted to find an almost minimum sized ε-net. A related problem that would be of great practical interest is to find a similar algorithm for ε-approximations. It is not even known if finding a minimum sized approximation is NP-complete, much less if there is any algorithm that returns an almost optimal ε-approximation.

The greedy method is shown to attain an $O(\log n)$ approximation ratio in the polytope separation, but it might be possible to use the geometry to show that it in fact yields a constant ratio. Such a claim has neither been proved nor disproved.

Finally, in the decision tree problem, we were unable to find a strategy that would work for degenerate arrangements. Is such an adaptation possible?

Acknowledgements

We would like to thank Dan Boneh, Bernard Chazelle, S. Rao Kosaraju, Jiří Matoušek, Joseph Mitchell, and Neal Young for helpful discussions concerning the topics of this paper.

References

[1] E. M. Arkin, H. Meijer, J. S. B. Mitchell, D. Rappaport, and S. S. Skiena. Decision trees for geometric models. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 369–378, 1993.

[2] P. Assouad. Densité et dimension. *Ann. Institut Fourier, Grenoble*, 3:232–282, 1983.

[3] E. Baum. On learning the union of halfspaces. *J. Compl.*, 6:67–101, 1990.

[4] M. Bellare, S. Goldwasser, C. Lund, and A. Russel. Efficient probabilistically checkable proofs and applications to approximation. In *Proc. 25th Annu. Symp. Theory Comput.*, pages 294–304, 1993.

[5] B. Berger, J. Rompel, and P. W. Shor. Efficient NC algorithms for set cover with applications to learning and geometry. In *Proc. 30th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 54–59, 1989.

[6] A. Blum and R. Rivest. Training a 3-node neural network is *np*-complete. In *Proc. 1st Workshop Comput. Learning Th.*, pages 9–18, 1988.

[7] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension. *J. ACM*, 36:929–965, 1989.

[8] H. Brönnimann. An implementation of MIN HITTING SET heuristics. Manuscript, 1994.

[9] Hervé Brönnimann, Bernard Chazelle, and Jiří Matoušek. Product range spaces, sensitive sampling, and derandomization. In *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS 93)*, pages 400–409, 1993.

[10] B. Chazelle. An optimal convex hull algorithm and new results on cuttings. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 29–38, 1991.

[11] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.

[12] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, and M. Sharir. Improved bounds on weak ϵ-nets for convex sets. In *Proc. 25th Annu. ACM Sympos. Theory Comput. (STOC 93)*, pages 495–504, 1993.

[13] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[14] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 281–290, 1993.

[15] B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete Comput. Geom.*, 4:467–489, 1989.

[16] V. Chvatal. A greedy heuristic for the set-covering problem. *Math. Oper. Res.*, 4:233–235, 1979.

[17] K. L. Clarkson. A Las Vegas algorithm for linear programming when the dimension is small. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 452–456, 1988.

[18] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[19] Kenneth L. Clarkson. Algorithms for polytope covering and approximation. In *Proc. 3rd Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes in Computer Science*, pages 246–252, 1993.

[20] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, Cambridge, Mass., 1990.

[21] G. Das. *Approximation schemes in computational geometry.* Ph.D. thesis, University of Wisconsin, 1990.

[22] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, New York, NY, 1979.

[23] M. T. Goodrich. Geometric partitioning made easier, even in parallel. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 73–82, 1993.

[24] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[25] D. Hochbaum. Approximation algorithms for the set covering and vertex cover problems. *SIAM J. Comput.*, 11(3):555–556, 1982.

[26] D. S. Hochbaum and W. Maas. Approximation schemes for covering and packing problems in image processing and vlsi. *J. ACM*, 32:130–136, 1985.

[27] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9:256–278, 1974.

[28] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[29] J. Komlós, J. Pach, and G. Woeginger. Almost tight bounds for ϵ-nets. *Discrete Comput. Geom.*, 7:163–173, 1992.

[30] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithms. In *Proc. 28th IEEE Sympos. Found. of Comput. Sci.*, pages 68–77, 1987.

[31] L. Lovász. On the ratio of optimal integral and fractional covers. *Discrete Math.*, 13:383–390, 1975.

[32] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. In *Proc. 25th Annu. Symp. Theory Comput.*, pages 286–293, 1993.

[33] J. Matoušek. Construction of ϵ-nets. *Discrete Comput. Geom.*, 5:427–448, 1990.

[34] J. Matoušek. Approximations and optimal geometric divide-and-conquer. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 505–511, 1991. Also to appear in *J. Comput. Syst. Sci.*

[35] J. Matoušek. Cutting hyperplane arrangements. *Discrete Comput. Geom.*, 6:385–406, 1991.

[36] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 276–285, 1992.

[37] J. Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.*, 2(3):169–186, 1992.

[38] J. Matoušek, R. Seidel, and E. Welzl. How to net a lot with little: small ϵ-nets for disks and halfspaces. In *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pages 16–22, 1990.

[39] J. Matoušek, E. Welzl, and L. Wernisch. Discrepancy and ϵ-approximations for bounded VC-dimension. In *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, pages 424–430, 1991.

[40] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[41] J. S. B. Mitchell and S. Suri. Separation and approximation of polyhedral surfaces. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 296–306, 1992.

[42] N. Sauer. On the densities of families of sets. *Journal of Combinatorial Theory*, 13:145–147, 1972.

[43] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory Probab. Appl.*, 16:264–280, 1971.

[44] E. Welzl. Partition trees for triangle counting and other range searching problems. In *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pages 23–33, 1988.

An Efficient Algorithm for the Euclidean Two-Center Problem

Jerzy W. Jaromczyk* Mirosław Kowaluk†

Abstract

We present a new algorithm for the two-center problem: "Given a set S of n points in the real plane, find two closed discs whose union contains all of the points and the radius of the larger disc is minimized." An almost quadratic $O(n^2 \log n)$ solution is given. The previously best known algorithms for the two-center problem have time complexity $O(n^2 \log^3 n)$. The solution is based on a new geometric characterization of the optimal discs and on a searching scheme with so-called lazy evaluation. The algorithm is simple and does not assume general position of the input points. The importance of the problem is known in various practical applications including transportation, station placement, and facility location.

1 Introduction

We study the following geometric optimization problem (the "two-center problem"): "Given a set S of n points in the real plane, find two closed discs whose union contains all of the points and the radius of the larger disc is minimized." The distance is measured with the Euclidean metric.

This is an instance of a well-known k-center problem, where the objective is to find k closed discs minimizing the maximum radius. The problem stems from practice. For example, we want to place k emergency units so the worst-case response time to each of n given sites will be minimized. The problem, specifically for $k = 1$, is known under various names: minimum spanning circle, minimax location problem, minimax detection station problem, Euclidean messenger boy problem (see [F1, F2, HM]). These names tell more about potential applications.

In addition to having a practical flavor the problem is known for its stimulating impact on inventing new algorithmic techniques. Since the problem was posed by Sylvester in 1857 [S] several algorithms for $k = 1$ have been presented. Recent computationally-efficient solutions include an $O(n \log n)$ time algorithm based on the farthest Voronoi diagram (see Shamos and Hoey [SH]). An optimal $O(n)$ algorithm based on a powerful, and beautiful, prune-and-search strategy was given by Megiddo [M]. The problem becomes NP-complete when the parame-

*Department of Computer Science, University of Kentucky, Lexington, KY 40506

†Institute of Informatics, Warsaw University, Warsaw, Poland

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

ter k is a part of the input. Therefore it is interesting, and again important in view of practical applications, to look at fixed values of k. Specifically, for $k = 2$ the problem was studied by Drezner [D] in the context of efficient transportation. He gave an $O(n^3)$ time complexity algorithm. Agarwal and Shamir [AS] provided the first algorithm with the cost close to quadratic. Specifically, their methods based on interesting modifications and additions to the Megiddo's parametric search technique resulted in a $O(n^2 \log^3 n)$ algorithm. An essential part of the algorithm is a decision procedure by Hershberger and Suri [HS]. This procedure, given a real number $r > 0$, determines in $O(n \log n)$ time if there are two closed discs of radius r whose union contains the given set of points. Moreover, it tells if the radius of two minimal enclosing discs is larger or smaller than r. Another algorithm with the same $O(n^2 \log^3 n)$ time complexity was presented by Katz and Sharir [KS] at last year's Symposium on Computational Geometry. Their paper brought out new ideas, based on expander graphs, for this and other geometric optimization problems. The complexity of the resulting algorithm for the two-center problem was the same $O(n^2 \log^3 n)$, however.

In this paper we will present a new algorithm for the two-center problem. The algorithm has $O(n^2 \log n)$ time complexity and is based on new observations regarding the geometry of the problem. They allow for an efficient search in the set of discs that are candidates for the optimal solution. The algorithm provides more insight into the geometry of the problem than the previous ones. (In fact, a lack of a direct geometric flavor is a known disadvantage of parametric-search based algorithms for geometric problems.) Due to this insight the resulting algorithm is simple, uniform, and works for arbitrary sets of points.

2 Geometric preliminaries

Our algorithm is based on a new characterization of the optimal discs. To proceed further we need some definitions. Assume that the plane is endowed with a standard orthogonal coordinate system. Consider a set of 12 unit length vectors that form with the axis OX an angle $\frac{i\pi}{6}$, $i = 0 \ldots 11$. Specifically, $u_i = [\cos((i\pi)/6), \sin((i\pi)/6)]$. With each point p in S we associate the set of 12 cones determined by the pairs of orthogonal vectors from this set of vectors. Each cone has its apex at p. Intuitively, we can think about this cone as the first quadrant (with the bounding half- lines) of the coordinate system centered at p and determined by vectors $[0,1], [1,0]$. All the other cones associated with p can be obtained by consecutive rotations of this quadrant by $\pi/6$ (30^o) about p. Clearly, the number of cones is $O(n)$.

Consider now the set S of n points. We say that a pair $(d^*, \bar{d}^*)$ of two closed discs is optimal for S if all the points in S are contained in the union $d^* \cup \bar{d}^*$, the larger of these two discs, say d^*, has radius r^* minimal over all pairs of discs whose union contains S, and $\bar{d}^*$ has the smallest possible radius for r^*. Disc d^* will be called *an optimal disc.* If $S_1 = \{p \in S : p \in d^*\}$ and $S_2 = S \setminus S_1$ then $\bar{d}^*$ is the minimal disc containing S_2. $\bar{d}^*$ will be called a *complementary* disc. Clearly, an optimal pair of discs is a solution to the two-center problem.

There are several simple and well-known properties of the two optimal circles. They will be useful through the rest of the paper and we will list them below.

Each disc in the optimal pair is determined either by a pair of points in S or by some three of them. That is, the radius of a disc is determined by half the distance between some two points, or is equal to the radius of the circumscribing circle for some three points in S. Note that, since we do not assume general position, the points determining the optimal discs are not unique. However, among the points determining the optimal discs there are points with special properties. We will formulate these for the case when both discs are defined by three points. The same will hold true for discs defined by two points.

There exist determining points A_1, B_1, C_1 for d^* that do not belong to the interior of $\bar{d}^*$, and there are A_2, B_2, C_2 that are not in the interior of d^*.

There exist points A_1, B_1, C_1 (and A_2, B_2, C_2, respectively) such that all angles in the triangle $\Delta A_1 B_1 C_1$ are acute. In other words, there are such A_1, B_1, C_1 that the center of their circumscribing circle is in the interior of $\Delta A_1 B_1 C_1$.

The above facts are elementary and follow from the observation that otherwise at least one of the discs, contrary to their optimality, could be made smaller. Hereafter, by the determining points for d^* and $\bar{d}^*$ we will mean points with the above properties. Note that one (or two) of the points determining d^* (the optimal disc) can lie on the boundary of the complementary disc $\bar{d}^*$ (but that this point does not determine $\bar{d}^*$ in the above sense).

From the above facts it is easy to derive that the radius of d^* is not bigger than the radius of the optimal spanning circle (the solution to the 1-center problem) for a given set. To this end, consider the optimal disc containing S and split S into two sets using a line containing the center of this disc and not containing any points in S. Each of these subsets can be enclosed in a smaller disc.

As opposed to the 1-center problem, which always has a unique solution, the 2-center problem can admit more than one optimal pair of discs. For example, the set of vertices of a square have two symmetric, optimal solutions.

Additionally to the above facts we will use a non-trivial result due to Megiddo [M] that provides a linear time algorithm to construct the minimal enclosing disc containing a given set of points. This algorithm will be useful for our purposes.

3 Ordered sequences of discs for cones

We will associate with each cone a fixed number of sequences of discs. Consider a cone C for a point p from S. Without loss of generality, we may assume that p is at the origin of the coordinate system and the cone coincides with the first quadrant of this system. Otherwise we can apply an affine transformation. Consider $S_C = S \cap C$ and its convex hull $CH(S_C)$, and also consider the set of discs that pass through p, that contain $CH(S_C)$, and whose circles contain at least one point in S_C beyond p. The locus of their centers is the boundary γ of the intersection of those halfplanes determined by the bisectors of the segments joining p with the vertices of $CH(S_C)$ that do not contain p. This intersection, and the boundary, can be found with a standard $O(n \log n)$ algorithm (see [E]). Another way to see this is to view γ as the envelope of the set of the above described bisector with the view point at $(+\infty, 0)$. If the order of the segments joining p with the vertices of $CH(S_C)$ is known (that is, the order of the slopes of bisectors is known) then this envelope can be found in $O(n)$ time with a modification of Keil's algorithm for the envelope of lines [K] (see also [V]).

γ is monotone with respect to the y-axis. This follows from the fact that all bisectors of the segments for points in the first quadrant are monotonic and that the intersection of halfplanes is a convex set.

Now consider the points $q_1, \ldots, q_m$ of S that are in the second and fourth quadrants of the coordinate system (the quadrants are numbered in the counterclockwise orientation, with the cone corresponding to the first quadrant). Each point q in these quadrants determines the minimal disc, with respect to the radius, whose circle passes through p and q and which contains $CH(S_C)$. It defines a sequence of discs $d(q_{i_1}), \ldots, d(q_{i_m})$ and the corresponding sequence of radii $r_{i_1}, \ldots, r_{i_m}$. We will show how the order of these radii can be found efficiently. We wish to do so without explicitly finding the discs; this would kill our hopes for an efficient algorithm. Before showing the method, let us explain at this point what our goal is. This ordered sequence of radii will allow us to perform a binary search provided that some kind of order can simultaneously be obtained for the complementary discs (the disc for the complement of S with respect to the points contained in the first disc.) Consider a point q and the segment pq. The center of the disc with the circle passing through p and q is located on the perpendicular bisector l_{pq} of pq. In order to find the center of the smallest such circle that in addition contains

$CH(S_C)$ we need to consider two cases that depend on how the middle point M of the segment pq is located with respect to γ. Its location depends on whether the disc that we consider is determined by three or by two points. If M is on that side of γ that contains p then the center is on the intersection of the bisector with γ. This intersection can be located, with the help of binary search, in $O(\log n)$ time. The radius of the disc is equal to the distance of this intersection to p. Otherwise, if M is on the opposite side of γ from p then M is the required center. Again, the radius is equal to the distance from M to p. Clearly, M can be located with respect to γ in $O(\log n)$ time. We put the centers of discs based on triples of points or based on pairs of points in different sequences. Sorting the distances yields the increasing order of the discs. Concluding, we have:

LEMMA **3.1** *The increasing order of the discs associated with the cone and determined by the points in the second quadrant can be found in $O(n \log n)$ time.*

Similarly, we can find the order of discs determined by the points in the fourth quadrant. Note that these centers of discs that are on γ do not need to be sorted since the distance of points on γ to p is a monotonic (unimodal, to be precise) function. The smallest radius corresponds to the minimal enclosing disc d_0 for $CH(S_C)$ and it splits γ into two monotonic subsequences corresponding to the second and the fourth quadrants respectively. We will consider each such part of the sequence separately. Note that the above construction works also if, instead of considering the set of all the points in the second and fourth quadrants, we consider only their subsets.

4 Monotone sequences of radii

A cone C_p associated with p determines an orthogonal coordinate system centered at p with the axes corresponding to the vectors defining C_p. The first quadrant $quadrant_1$ coincides with the cone. Let $quadrant_2$, $quadrant_3$, and $quadrant_4$ be the remaining quadrants (in counterclockwise order). We will look at the points $Q_i = S \cap quadrant_i$ (the first quadrant contains the positive axis, i.e. is closed, the negative x-axis belongs to $quadrant_2$, and the negative y-axis belongs to $quadrant_4$).

As we know, C_p together with Q_2 determines a sequence of discs ordered with respect to non-decreasing radii. (Similarly, another sequence is defined by C_p with Q_4.) Instead of constructing the sequence for all points in Q_2 (Q_4) we will construct it only for those points in Q_2 (Q_4, respectively) that are not in disc d_0 which is the minimum enclosing disc for Q_1. Ignoring those points will not affect the correctness of the overall algorithm for the 2-center problem. As we will show later, points in the second and fourth quadrants will be examined as potential candidates for the point that determines, together with the apex of the cone and another point in $CH(S_C)$ an optimal circle. It follows from the definition of the sequences and from simple geometric facts that points that are inside d_0 are not feasible candidates since they could not form an acute triangle with the cone center and another point in $CH(S_C)$ and simultaneosly pass through the cone center and contain all points in $CH(S_C)$. Denote the obtained sequence by $(d_i)_0^{k_C} = (d_0, d_1, \ldots, d_{k_C})$, the corresponding radii by $r(d_i)$, and the set of points of S in d_i by $d_i(S)$. The sequence starts with disc d_0, the minimum enclosing set for Q_1 that can be found in $O(n)$ time with Megiddo's algorithm [M]. Others of these sets are not explicitly known; they can be found with an additional $O(n)$ time per set, but this is too costly an operation for us so we do not intend to compute all of them. Each d_i determines its complementary disc $\bar{d}_i$ which is the minimal enclosing disc for the complement $\bar{d}_i(S)$ of $d_i(S)$, i.e. for $S \setminus d_i(S)$. Denote by $\bar{r}_i$ the radius of $\bar{d}_i$. It gives us a sequence of pairs of numbers $\{(r_i, \bar{r}_i)\}$. Let us emphasize that the first elements of the pairs are known from the algorithm constructing the sequence (see the previous section). However, it requires an extra effort (e.g. running Megiddo's algorithm), which we want to avoid whenever possible, to find $\bar{r}_i$. Direct search of the sequence for the optimal pair of discs (passing through p) is not effective since $(\bar{r}_i)$

is not monotonic in general. This stems from the fact that, in general, the sequence of sets $d_i(S)$ is not monotone with respect to inclusion. Before proceeding further, in order to see where the difficulty lies, let us analyze what happens to the sets $d_i(S)$ of points in the sequence (growing with respect to the radius) of discs associated with the cone and Q_2. To focus our attention, let us consider those sequences with discs whose circles pass through at least one other point of S in the cone beyond p. The growing discs contain more and more points (in the sense of set inclusion) from Q_2 and Q_3. This follows from the observation that their circles must intersect in Q_1 in order to contain S_1. Therefore, there is no intersection in the third quadrant; this also translates into the fact that all points in Q_2 (and Q_3) that are in the smaller disc must be also in the larger disc. By the same argument, the discs can loose points from Q_4 (the fourth quadrant) as they grow. Because of these points in Q_4, the monotonicity (w.r.t. inclusion) of $d_i(S)$, and therefore the monotonicity of $S \setminus d_i(S)$, can be lost. We will soon see that there are yet some cones that give us control over the events occurring in the fourth quadrant. We now have the following theorem, formulated here for the case of the optimal disc defined by some three points. The case of discs defined by some two points is similar and will be discussed later.

THEOREM **4.1** *Let d^* be an optimal disc (the larger of the two discs in an optimal pair) whose circle passes through p. There exists a cone C_p for p such that $(S \cap C_p) \subset d^*(S)$ and one of the following holds true:*

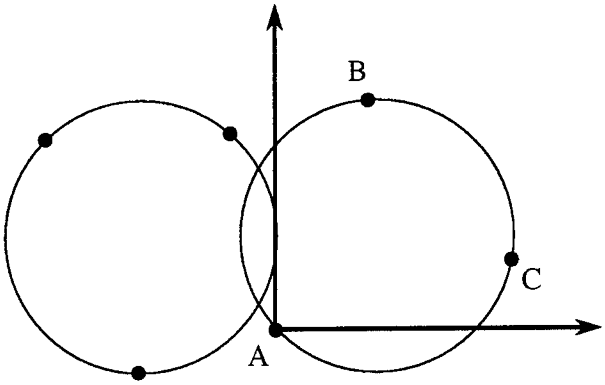
1. *All three points determining d^* are in C_p (p being one of them)*

2. *C_p contains two points determining d^* (p being one of them), and no points of S that are in the optimal disc are in Q_4 (or, symmetrically, in Q_2). That is, $d^* \cap Q_4 = \emptyset$.*

3. *C_p contains two points determining d^* (p being one of them), and the first disc d_1 (not d_0) contains all the points in Q_4 (or in Q_2, respectively) that are in d^*. That is, $(d^*(S) \cap Q_4) \subset d_1(S)$ (or, $(d^*(S) \cap Q_2) \subset d_1(S)$), respectively.*

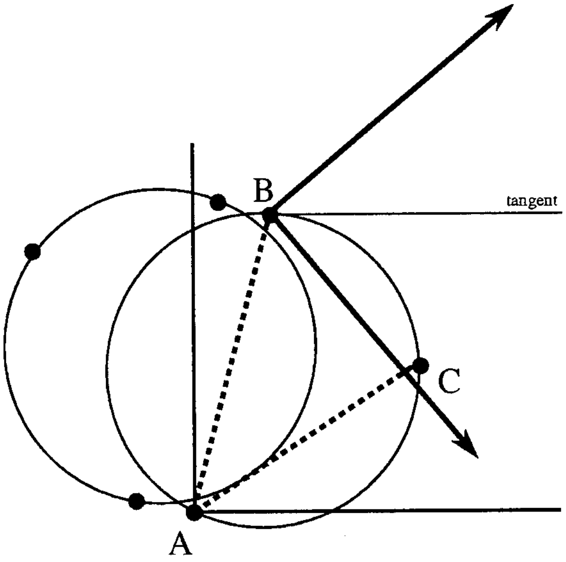
PROOF: Let A, B, C be points that determine d^*. Let the projection of A on the line joining the centers of d^* and $\bar{d}^*$ be the closest to the center of $\bar{d}^*$, and let the projection of C be the farthest (with ties broken arbitrarily). We will carry out the proof under the assumption that A is "below" the line joining the centers of the discs, and that we rotate cones in the clockwise direction. Other cases are symmetric. We have the following three possibilities.

1. There exists a cone with $p = A$ that contains both B and C such that no point in $S \setminus d^*(S)$ belongs to this cone. Then the minimal disk enclosing the points in this cone (and passing through A), that is d_0, is an optimal disk.



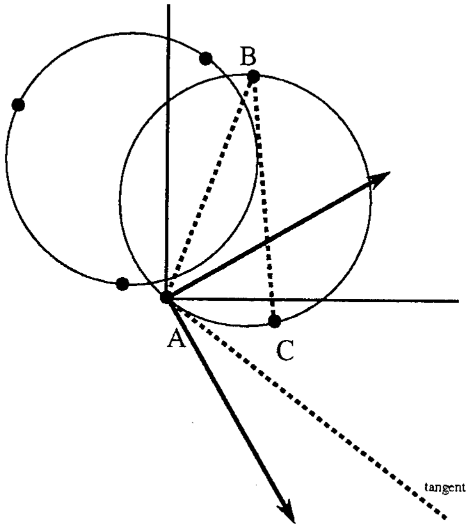
2. The above case does not hold true and one of the angles BAC or ABC is smaller than $\pi/6$ (30^o). We will consider both cases.

a) $BAC < \pi/6$.



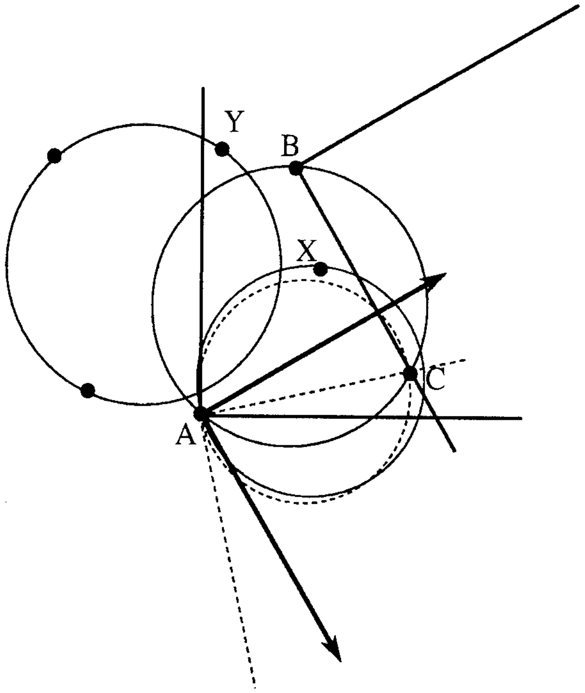
Take the cone for A that contains B and such that the next cone does not contain B. This cone contains some points in $S \setminus d^*(S)$. Now take the similarly oriented cone C_B for B and consider the cone second next to it (i.e. rotate the cone about B by $\pi/3$.) We will show that this cone has the desired properties. This cone determines four quadrants, as described earlier. First, note that C belongs to this cone (the first quadrant) since the angle ABC is at least $\pi/3$, and the angle between AB and one of the arms of C_B is not smaller than $\pi/2$ and not bigger than $2\pi/3$. Note that no point of S in the optimal disc belongs to the fourth quadrant. This follows from the fact that angle ABO, O being the center of d^*, is smaller than $\pi/6$ and therefore a half line of the tangent line to d^* at B is in the first quadrant. Also, no point in $S \setminus d^*(S)$ is in the first quadrant (the cone) since it lies completely in the halfplane determined by the perpendicular to the segment joining the centers of d^*, and $\bar{d}^*$ through point B. Hence, the cone has the required properties.

b) $ABC < \pi/6$.



Take the cone for A that contains B and such that the next cone does not contain B. Now, consider the cone second next to it (i.e. rotate the cone about A by $\pi/3$.) We will show that this cone has the desired properties. This cone determines four quadrants, as described earlier. C belongs to this cone (the first quadrant) since the angle BAC is smaller than $\pi/2$ and bigger than $\pi/3$. Note that no point of S in the optimal disc belongs to second quadrant. This follows from the fact that angle BAO, O being the center of d^*, is smaller than $\pi/6$ and therefore a half line of the tangent line to d^* at A is in the first quadrant. Also, no point in $S \setminus d^*(S)$ is in the first quadrant (the cone) since it lies completely in the halfplane determined by the perpendicular to the segment joining the centers of d^*, and $\bar{d}^*$ through point A (which follows from the fact that angle ABC is less than $\pi/6$). Hence, the cone has the required properties.

3. The first case does not hold and both the angles ABC and BAC are at least $\pi/6$. Note that at least one of the right angles centered at A or B, with one arm passing through C and the other not intersecting d^*, does not contain points in $S \setminus d^*(S)$. Assume that it is the right angle centered at A; denote it by R_A. Take the cone determined by A that contains C and the next one that does not have this property. We will show that this cone has the desired properties. First note that the first quadrant does not contain points in $S \setminus d^*(S)$.



Those points that belong to $d^*(S)$ and are in the fourth quadrant are contained in the disc d_{AC} with the diameter AC. Furthermore, the intersection of the first disc d_1 in the sequence generated by the second quadrant with the fourth quadrant does not contain any point in $S \setminus d^*(S)$ as it is completely included in right angle R_A. This follows from the fact that the circle of d_1

intersects the circle of d_{AC} at A and in the arc AC. This intersection also contains the intersection of d^* and the fourth quadrant Q_4. Therefore, d_1 contains all the points of S that are in $d^*(S) \cap Q_4$. Hence, the cone has the required properties. □

The immediate consequences of the above theorem are:

COROLLARY **4.1** **1.** *The optimal disc is defined by the minimum spanning disc of the points in some cone; or*

2. *the optimal disc is in a trailing segment* $d_s, \ldots, d_{k_C}$ *of the sequence of discs* $d_0, d_1, \ldots, d_{k_C}$ *for some cone. This segment consists of those* d_i*'s that do not contain any points of* Q_4*. Hence, we have* $d_s(S) \subset \ldots \subset d_{k_C}(S)$*; or*

3. *the optimal disc is in an initial segment* $d_1, \ldots, d_{k'_C}$ *of the sequence* $d_1, \ldots, d_{k_C}$ *for some cone* C_p*. (Note that we start this sequence at* d_1 *rather than at* d_0*). This initial segment consists of those* d_i*'s that contain the same points of* Q_4 *(or* Q_2*) as disc* d_1 *does. Hence, we have* $d_1(S) \subset \ldots \subset d_{k'_C}(S)$.

Note that the segments of the sequences from items 2 and 3 of the above corollary can be found from the sequence of the discs in $O(n \log n)$ time using a simple binary search. In later sections we will always assume that the sequences of discs are shortened to appropriate segments, as described above.

Recall that the monotonicity (w.r.t. set inclusion) of $d_i(S)$ implies the monotonicity (in the opposite direction) of the complements $\bar{d}_i(S) = S \setminus d_i(S)$. This, in turn, implies the monotonicity of the corresponding radii $\bar{r}_i$. This holds only for some cones, as described in the theorem. Nevertheless, we will soon see that it is sufficient to find the optimal discs efficiently.

We need to comment why the third quadrant is ignored in building sequences of candidate discs. Clearly points in the third quadrant can belong to d^*. However, no point in the third quadrant can determine an optimal disc together with p (the center of the cone) and a point in the first quadrant as they form an obtuse triangle.

We will end this section with results for an optimal disc that is determined by a pair rather than a triple of points. In this case the segment joining this pair of points is a diameter of the disc.

THEOREM **4.2** *Let* d^* *be an optimal disc whose circle contains point* $p \in S$*. There exists a cone* C_p *for* p *such that* $S \cap C_p \subset d^*(S)$ *and one of the following holds true:*
1. Both point determining d^* *(*p *being one of them) belong to* C_p*; or*
2. The other than p *point determining* d^* *is not in* C_p*. Then* $d^*(S) \cap Q_4 = \emptyset$ *or* $d^*(S) \cap Q_2 = \emptyset$.

PROOF: Let A and B be the points determining d^*. Let A be this point whose projection on the line joining the centers of d^* and $\bar{d}^*$ is closer to the center of $\bar{d}^*$ than the projection of C. We will carry out the proof under the assumption that A is "below" the line joining the centers of the discs, and that we rotate cones in the clockwise direction. Other cases are symmetric. Consider this cone C_A that does not contain any points in $S \setminus d^*(S)$ and the previous one does not have this property. Such a cone always exists as there is a halfplane determined by a line passing through A that do not contain any points in $S \setminus d^*(S)$. C_A determines four quadrants as usual, the cone itself being the first one. We have two cases:
1. B belongs to C_A. Then the cone has the required properties.
2. B is not in C_A. Assume that B is in Q_2 (the second quadrant). Then the tangent line to d^* at A passes through the first and third quadrants and therefore no point in $d^*(S)$ belongs to Q_4. □

Note that the above proof is significantly simpler than its counterpart for discs determined by triples of points. One of the reasons is that we do not need to find a cone that necessarily contains the second determining point (beyond p) in its interior as it is required in the case of triples.

To gain a benefit from the above theorem we need another straightforward observation.

If d^* is an optimal circle determined by a pair of points p and q (with p's projection on the segment joining the centers of d^* and $\bar{d}^*$ being closer to the center of $\bar{d}^*$) then at least one of the right angles centered at p with one arm including pq does not contain points in $S \setminus d^*(S)$. This follows from the fact that p does not belong to the interior of $\bar{d}^*$ so $\bar{d}^*$ can not overlap these both right angles at the same time.

Now, the above theorem leads to

COROLLARY **4.2** *d^* is either identical with d_0 or it belongs to the sequence of discs $d_1, d_2, \ldots, d_C$ as defined in Section 2.*

4.1 Searching with lazy evaluations

The searching in the algorithm will be based on the following general scheme. Assume that we are given a family Ω of sequences of pairs of numbers $\{(a^i_j, b^i_j)\}$, $j = 1 \ldots m_i$, $i = 1 \ldots k$, and set $m = \sum_1^k m_i$. Denote by Ω^* the set of all pairs $(a, b) \in \Omega$ such that $b \leq a$. The searching problem is: "Find a lexicographically smallest pair $(a^*, b^*) \in \Omega^*$." That is, we want to find a pair with the smallest a, b not greater than a, and with b as small as possible for a given a.

Assume that, in addition, the sequences in Ω have the following properties. There exists i_0 such that $\{a^{i_0}_j, b^{i_0}_j\}$ contains (a^*, b^*); $\{a^{i_0}_j\}$ is a nondecreasing sequence; and $\{b^{i_0}_j\}$ is a nonincreasing sequence. In other words, the optimal (a^*, b^*) is contained in some sequence of pairs whose first elements form a monotonic sequence, and whose second elements also form a monotonic sequence (in the opposite direction). The collection Ω with the above properties will be called *quasi-monotonic.*

We are interested in the case when the a's are known and the b's can be computed with a certain cost penalty. To make the discussion more specific assume that the number of sequences $k = O(n)$, the lengths of the sequences are $O(n)$, and the cost to find each b is also $O(n)$. In this case a standard search for the maximum would result in $O(n^3)$ time. Due to the special properties of quasi-monotonic families we can perform that search more efficiently.

The following simple binary-like search can efficiently locate (a^*, b^*). For each sequence perform a binary search with respect to the a's in order to find the smallest a with $b \leq a$ (a local minimum). If the a's form a nondecreasing and the b's a nonincreasing sequence, a binary search will correctly locate the required pair. However, not all sequences are monotonic with respect to the a's and b's, and therefore the search can lead to one of two possible outcomes. Here the search is guided by the result of comparing the current a with the corresponding b (that must be evaluated). For example, if $b > a$, the search will continue in the right half. Either we will not find any pair with $b \leq a$ and no pair will be returned or the search will continue and a pair will be returned. In this case the returned pair is not guaranteed to be the smallest in a given sequence. Now, from all the returned pairs we choose the lexicographically smallest. This must be our (a^*, b^*) since it is an element of at least one monotonic sequence; the binary search correctly locates it. The total cost of the search is bounded by $O(n^2 \log n)$. Concluding, we have:

THEOREM **4.3** *The lexicographically smallest pair (a^*, b^*), with $b \leq a$, of a quasi-monotonic collection of $O(n)$ sequences, with $O(n)$ pairs in the sequence and $O(n)$ penalty to evaluate pairs, can be found in $O(n^2\ logn)$ time.*

5 Algorithm

We will use the schema of searching with lazy evaluations for locating the optimal solution $(d^*, \bar{d}^*)$. We have the following lemma

LEMMA **5.1** *The family Ω of all the sequences associated with the cones is quasi $-$ monotone.*

PROOF: Follows directly from the construction of the sequences and from Corollary 4.1 and 4.2. □

The algorithm works as follows. For all the points in S construct the corresponding cones, and the sets Q_1, Q_2, Q_3, and Q_4. Find the minimal enclosing disc d_0 for Q_1 and the minimal disc $\bar{d}_0$ for $S \setminus d(S)$ and add it to the pool of candidates. Then construct ordered sequences of discs for each cone as described in Section 2. This takes $O(n \log n)$ time per cone and $O(n^2 \log n)$ for all cones. From these sequences we can construct those sequences described in items 2 and 3 of Corollary 4.1 in an additional $O(n \log n)$ time per cone. For example, with the help of a binary search we can locate the last disc in the sequence that contains the same points of Q_4 as the first disc d_1 of this sequence does. Based on Theorem 4.1 and 4.2 the set of sequences contains an optimal disc. By virtue of Lemma 5.1 this set of sequences is quasi-monotone. Perform the lazy evaluations search on this set and return pairs of discs into the pool of candidates. This can be done by Theorem 4.3 in $O(n^2 \log n)$ time. The penalty for each lazy evaluation is $O(n)$ if Megiddo's algorithm is used to find the complementary discs. Finally, find the global minimum $(d^*, \bar{d}^*)$ (or all pairs of discs that are optimal) in $O(n)$ time. This leads to the following main result:

THEOREM **5.1** *The pair of discs $(d^*, \bar{d}^*)$ that solves the two-center problem for a set of n points in the Euclidean plane can be found in $O(n^2 \log n)$ time.*

6 Conclusions

We have presented a new algorithm for the two-center problem. Its cost of $O(n^2 \log n)$ is the best known to date. The algorithm is based on an interesting insight into the geometry of the problem that leads to a simple searching scheme called searching with lazy evaluations. The algorithm is robust with respect to degenerate configurations of points. It also handles uniformly optimal discs that are defined either by three or by two points. The searching scheme is simple and therefore it will be interesting to see whether or not there is some room for improvement in the time complexity. Interesting directions of further research include studies on more general k-center problems for a fixed value of k, and for non-Euclidean metrics. We hope that the geometric properties of the optimal discs presented in this paper will find more applications in geometric optimization.

Acknowledgement: We thank Richard King for his help in the proofreading of this paper and Dan Boylin for his help with preparing the figures.

Bibliography

[AS1] P. K. Agarwal and M. Sharir. Planar geometric location problems, Tech. Rep. 90-58, DIMACS, Rutgers Uni., August 1990. (also to appear in the *Algorithmica*)

[D] Z. Drezner. The planar two-center and two-median problems, *Transportation Science*, 18, 1984, pp.351-361.

[E] H. Edelsbrunner. Algorithms in Computational Geometry, Springer Verlag, 1987.

[F1] R. R. Francis. A note on the optimum location of new machines in existing plant layouts. *J. Industr. Eng.* 12, 1961, pp.41-47.

[F2] R. R. L. Francis. Some aspects of minimax location problem, Oper. Res., 15 (1967), pp.1163-1168.

[HM] G. Y. Handler and P. B. Mirchandani. Location on Networks: Theory and Algorithms, MIT Press, Cambridge, MA., 1979.

[HS] J. Hershberger and S. Suri. Finding Tailored Partitions, *J. of Algorithms* 12 (1991), pp. 431-463.

[K] M. Keil. A simple algorithm for determining the envelope of a set of lines, *Information Processing Letters*, 39 (1991), pp.121-124.

[KS] M. J. Katz and M. Sharir. An Expander-Based Approach to Geometric Optimization. Proc. of the ninth Symposium on Computational Geometry, ACM, pp. 198-207, 1993.

[M] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems, *SIAM J. Comp.*, 12 (1983), pp.759-776.

[RT] H. Rademacher and O. Toeplitz. The Enjoyment of Mathematics, Princeton Univ. Press, Princeton, NJ, 1957.

[S] J. J. Sylvester. A question in geometry of situations, Quart. J. Math. 1(1857), p.79.

[SH] M. I. Shamos and D. Hoey. Closest-Point Problems, Proc. FOCS, IEEE (1975), pp.151-162.

[V] G. Vegter, Computing the bounded region determined by finitely many lines in the plane, Tech. Report 87-03, Dept. Math. and Comp. Sci., Rijkuniversiteit, Groningen, 1987.

On geometric optimization with few violated constraints*

Jiří Matoušek

Department of Applied Mathematics
Charles University
Malostranské nám. 25, 118 00 Praha 1, Czech Republic

Abstract

We investigate the problem of finding the best solution satisfying all but k of the given constraints, for an abstract class of optimization problems introduced by Sharir and Welzl — the so-called LP-type problems. We give a general algorithm and discuss its efficient implementations for specific geometric problems. For instance, for the problem of computing the smallest circle enclosing all but k of the given n points in the plane, we obtain an $O(n \log n + k^3 n^\varepsilon)$ algorithm; this improves previous results for k small compared to n but moderately growing.

We also establish some results concerning general properties of LP-type problems.

1 Introduction

Smallest enclosing circles. We begin by discussing the following geometric problem:

> Given a set P of n points in the plane and an integer q, find the smallest circle enclosing at least q points of P.

This problem has recently been investigated in several papers [AIKS91], [EE93], [DLSS93], [ESZ93], [Mat93b] and it motivated also this paper. In these previous works, algorithms were obtained with roughly $O(nq)$ running time[1]. There seems to be little hope at present at improving these bounds substantially in the full range of values of q. In particular, as observed by D. Eppstein (private communication), for q close to $n/2$, a subquadratic solution would imply a subquadratic solution for several other basic problems, for which one seems unlikely with present methods (information about this class of problems, collected and publicized under the name 'n^2-hard problems' by M. Overmars and his coworkers, can be found in [GO93]; see also Erickson and Seidel [ES93] for lower bounds in a weak model of computation for some problems of this type).

In this paper we investigate improvements over the roughly $O(nq)$ bound in the situation when $k = n - q$ is small compared to n (all but few points should be enclosed by the circle); this question was raised e.g., by Efrat *et al.* [ESZ93]. One of the first methods coming to mind for solving the problem is to construct the qth order Voronoi diagram for the point set P, and find the required circle by inspecting all its cells (this approach was pointed out by Aggarwal *et al.* [AIKS91]). It is known that the combinatorial complexity of the qth order Voronoi diagram is $\Theta((n-q)q)$, and it has been shown recently that it can be constructed in expected time $O(n \log^3 n + (n-q)q \log n)$ [AdBMS94], (see also [Cla87], [AM91] for previous results). In our setting, this says that the smallest circle enclosing all but k points can be found in $O(n \log^3 n + nk \log n)$ time.

In this paper we show that one can do still better for small k, namely that the problem can be solved in close to linear time with k as large as $n^{1/3}$:

Theorem 1.1 *The smallest circle containing all but at most k of the given n points in the plane can be computed in $O(n \log n + k^3 n^\varepsilon)$ time*[2].

*Supported in part by Charles University grant No. 351 and Czech Republic Grant GAČR 201/93/2167. Part of this research was performed while the author was visiting at Computer Science Institute, Free University Berlin, and it was supported by the German-Israeli Foundation of Scientific Research and Development (G.I.F.), and part while visiting Max-Planck Institute for Computer Science in Saarbrücken.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

[1] Here are the specific running times: Eppstein and Erickson [EE93] solve the problem in $O(nq \log q + n \log n)$ time with $O(n \log n + nq + q^2 \log q)$ space, and Datta *et al.* [DLSS93] give an algorithm with the same running time and space improved to $O(n + q^2 \log q)$. Efrat *et al.* [ESZ93] achieve $O(nq \log^2 n)$ time with $O(nq)$ space or alternatively $O(nq \log^2 n \log(n/q))$ time with $O(n \log n)$ space, and the author [Mat93b] has $O(n \log n + nq)$ time with $O(nq)$ space or time $O(n \log n + nq \log q)$ with $O(n)$ space.

[2] Throughout the paper, ε in exponents stands for a positive constant which can be made arbitrarily small by adjusting the

A predecessor of our technique is a result of Megiddo [Meg83], who demonstrated that the $k = 0$ case, the smallest enclosing circle for n points, can be solved in $O(n)$ time.

LP-type problems. The problem of finding the smallest enclosing circle belongs to a class of optimization problems known as LP-type problems (or 'generalized linear programming' problems). This class was introduced by Sharir and Welzl [SW92]. It captures the properties of linear programming relevant for the description and analysis of their linear programming algorithm. The definition and more information on LP-type problems will be given later; here we only mention few important facts.

Each LP-type problem has an associated parameter, the so-called combinatorial dimension (or dimension for short). For instance, for a feasible linear programming problem (one with a solution satisfying all the constraints), this combinatorial dimension equals to the geometric dimension of the underlying space. Randomized algorithms are known for solving an LP-type problem with n constraints and of dimension bounded by a constant in $O(n)$ expected time, see [Cla88], [SW92]. For a linear-time deterministic algorithm under somewhat more restrictive assumptions see [CM93]. The algorithms have originally been developed for linear programming, but under suitable computational assumptions they work for all LP-type problems. At the same time, the class of LP-type problems turned out to contain also many other optimization problems of geometric flavor besides linear programming, such as the small enclosing circle computation or finding the distance of two convex polyhedra. See also [Kal92], [MSW92] for algorithms with expected running time subexponential in the dimension, [Gär92] for another but related abstract class of optimization problems and [Ame93] for further properties and applications of the LP-type class.

The problem of finding the best solution satisfying all but at most k of the given constraints can be posed and solved within the framework of LP-type problems. We need that the considered LP-type problem is *nondegenerate*. In geometric situations, this roughly means that the constraints are in general position, which can usually be achieved by perturbation techniques similar to the ones for removing degeneracies in geometric problems. However, one has to proceed carefully so that the combinatorial structure of the problem is not changed in an undesirable way. Exact definitions for the notions in the following theorem are given in the next section.

Theorem 1.2 *Let d be a constant, let (H, w) be a nondegenerate LP-type problem of dimension d with n constraints and let k be a given integer, $1 \leq k < n$. Then the minimum basis violated by at most k of the constraints can be computed in $O(nk^d)$ time, by finding optima for $O(k^d)$ LP-type problems of the form (G, w) with $G \subseteq H$.*

Paper overview and further results. In section 2, we first review the definitions and some properties for LP-type problems (sec. 2.1). Then we define and discuss nondegenerate LP-type problems (section 2.2). In section 2.3 we consider bounds on the number of bases violated by at most k constraints and by exactly k constraints, phrasing results and proofs known for linear programming in the LP-type problems setting. In section 2.4 we give a simple algorithm for finding the minimum basis violated by at most k constraints in a nondegenerate LP-type problem, and we also show that our notion of "removing degeneracies" is meaningful in this context.

The general algorithm can sometimes be implemented more efficiently in specific geometric situations using dynamic data structures. Theorem 1.1 is one such case. In section 3, we discuss this and some other geometric problems, mainly the following one (derived from linear programming):

Problem 1.3 *Given a collection H of n closed halfspaces in $\mathbb{R}^d$ and an integer k, $1 \leq k < n$, find the lexicographically smallest point of $\mathbb{R}^d$ contained in all but at most k of the halfspaces of H.*

Other geometric problems amenable to a similar treatment will be only mentioned without going into details.

After we deal with technical obstructions concerning degeneracies, Problem 1.3 can be solved using the general algorithm from Theorem 1.2. There are two substantially different cases: if the linear programming problem defined by H is feasible (that is, the intersection of all halfspaces of H is nonempty), the dimension of the corresponding LP-type problem is d, while for an infeasible problem the dimension is $d+1$ (note that "feasible", "infeasible" refers to the existence of a point in *all* halfspaces of H, not to the existence of a point lying in all but at most k halfspaces of H).

By using dynamic data structures, the complexity of the general algorithm can be improved. The results are summarized in the following theorem (we do not write out the improved complexities for larger dimensions, as the improvements become less significant):

Theorem 1.4 (i) *In the* feasible case, *Problem 1.3 can be solved*

- *in time $O(nk^d)$ by the general algorithm, for any fixed d,*
- *in time $O(n \log n + k^3 n^\varepsilon)$ in dimension 3, and*

parameters of the algorithms. Multiplicative constants implicit in the $O()$ notation may depend on ε.

- *in time $O(n \log n + k^{8/3}n^{2/3+\varepsilon} + k^4 n^{1/3+\varepsilon})$ in dimension 4.*

(ii) *In the* infeasible case*, we get*

- *$O(nk^{d+1})$ time for the general algorithm,*
- *$O(n \log n + k^3 \log^2 n)$ time in dimension 2 and*
- *$O(n \log n + k^4 n^\varepsilon)$ time in dimension 3.*

Let us summarize previous work on Problem 1.3. In the *feasible case*, the problem can be rephrased as finding the lexicographically smallest point of the k-level in an arrangement of n hyperplanes.

In the plane, there is an $O(n \log^2 n)$ algorithm (this is why we do not mention the planar case in Theorem 1.4). It is based on the observation that for a given horizontal line, one can determine in $O(n \log n)$ time whether it intersects the k-level. Then parametric search is used to determine the lowest line which still intersects the k-level. This algorithm (implicitly) appears in Cole *et al.* [CSY87]. It is not clear at present how efficiently can this approach be generalized to higher dimensions.

A natural approach in higher dimension is to construct the k-level or the $(\leq k)$-level in the arrangement[3]. In dimension 3, an output-sensitive algorithm is known which constructs the k-level in $O(n^\varepsilon(b + n))$ time, where b denotes the complexity of the constructed level [AM91]. The worst-case complexity of the k-level is $O(n^{8/3})$ [DE93] (see also [ACE+91]). In higher dimensions, only weak worst-case bounds on the k-level complexity are known, and also output-sensitive construction algorithms become less effective, so that one may consider constructing the whole $(\leq k)$-level. This can be done in expected $O(nk^2 \log \frac{n}{k})$ time in dimension 3 and in worst-case optimal $O(n^{\lfloor d/2 \rfloor}k^{\lceil d/2 \rceil})$ expected time in dimension $d \geq 4$ [Mul91]. For k much smaller than n, the level construction approach thus becomes quite inefficient in higher dimensions. Let us remark that the picture is very different for "random" problem instances, as Mulmuley [Mul91] proves that for hyperplanes randomly chosen from several natural distributions, the $(\leq k)$-level expected complexity as well as expected construction time are $O(nk^{d-1})$.

Problem 1.3 in the *infeasible case* was investigated in several papers. For instance, Cole *et al.* [CSY87] and Everett *et al.* [ERvK93] considered the following *weak separation* problem in the plane: Given two point sets R and B with n points in total, find the smallest k for which there is a line ℓ and a k-point set $E \subseteq R \cup B$ such that all points of $R \setminus E$ lie on one side of ℓ and all point of $B \setminus E$ lie on the other side. In a dual setting, this amounts to finding the smallest k for which there exists a point in the plane contained in all but k of the given halfplanes. [ERvK93] provide an $O(nk \log k + n \log n)$ solution, based on a $(\leq k)$-level construction algorithm. Efrat *et al.* [ELS93] consider the problem in the latter form and obtain similar results. Moreover, they mention some more applications and investigate the analogous problem in dimension 3, where they give an $O(nk^2(\log n + \log^2 \frac{n}{k}))$ solution. Another closely related problem is the finding of a line intersecting all but at most k of given n vertical segments in the plane, investigated by Everett *et al.* [ERvK93]; it can also be reduced to linear programming with at most k violated constraints.

Our results for both the 2 and 3-dimensional problem thus present an improvement over the previous roughly $O(nk)$ bounds if k is in the range (roughly) from $\log n$ to $\sqrt{n}$. Let us remark that the smallest k (in the weak separation problem or its dual) can be found in the same time as are the bounds in Theorem 1.4 (where the k is given), since our method searches all bases of level at most k, and we may arrange it so that bases of smaller level are searched first.

2 LP-type problems

2.1 Basic definitions and properties

We begin by recalling the abstract framework of [SW92] (with minor formal modifications). An *minimization problem* is a pair (H, w), where H is a finite set, and $w : 2^H \longrightarrow \mathcal{W}$ is a function with values in a linearly ordered set $(\mathcal{W}, \leq)$. The elements of H are called the *constraints*, and for a subset $G \subseteq H$, $w(G)$ is called the *value* of G.

Intuitively, the value $w(G)$ for a subset G of constraints stands for the smallest value attainable for certain objective function while satisfying all the constraints of G. The goal is to find $w(H)$. For the computation, the problem is not specified by giving the value of w for each subset (which would make the computation trivial), but rather by oracles implementing certain primitive operations, to be described below.

The set $\mathcal{W}$ is assumed to posses a smallest element denoted by $-\infty$ (intuitively, it stands for 'optimum undefined') and usually also a largest element ∞ (with intuitive meaning 'no feasible solution exists').

The minimization problem (H, w) is called an *LP-type problem* if the following two axioms are satisfied:

> **Axiom 1.** (Monotonicity) For any F, G with $F \subseteq G \subseteq H$, $w(F) \leq w(G)$.
> **Axiom 2.** (Locality) For any $F \subseteq G \subseteq H$ with $w(F) = w(G) > -\infty$ and any $h \in H$, $w(G \cup \{h\}) > w(G)$ implies that also $w(F \cup \{h\}) > w(F)$.

Before we specify the computational primitives, let

[3] Even in the plane, the above mentioned $O(n \log^2 n)$ algorithm can be improved a little for k very small, see [ERvK93].

us introduce some more terminology. A *basis* B is a set of constraints with $w(B') < w(B)$ for all proper subsets B' of B. A *basis for* a subset G of H is a basis $B \subseteq G$ with $w(B) = w(G)$. So a basis for G is a minimal subset of G with the same value as G. The maximum cardinality of any basis is called the *dimension of* (H,w) and denoted by $\dim(H,w)$.

We say that a constraint $h \in H$ *violates* a set G if $w(G \cup \{h\}) > w(G)$. This notion is most often used with G being a basis. For $G \subseteq H$ we denote by $V(G)$ the set of constraints of H violating G.

As was mentioned in the introduction, there are several algorithms for solving an LP-type problem of constant bounded dimension in time $O(|H|)$. These algorithms differ slightly in the assumptions on the primitive operations available for the considered problem. We describe the primitives needed for the randomized algorithm of Sharir and Welzl [SW92] (see also [MSW92]).

> **Violation test.** Given a basis B and a constraint $h \in H$, decide whether h violates B.
>
> **Basis change.** Given a basis B and a constraint h, return (some) basis for $B \cup \{h\}$.
>
> **Initial basis.** On the beginning, we have some basis B_0 with $w(B_0) > -\infty$.

For the $O(nk^d)$ time bound in Theorem 1.2, we assume that both violation test and basis change are implemented in constant time, and that for any $G \subseteq H$ with $w(G) > -\infty$ an initial basis $B_0 \subseteq G$ with $w(B_0) > -\infty$ can be found in $O(n)$ time.

To illustrate the definitions, let us look how two particular problems fit into this framework. For the problem of finding the *smallest enclosing circle* for a set H of points in the plane, the constraints are the points, the value $w(G)$ of a nonempty set $G \subseteq H$ is the radius of the smallest circle enclosing G, and we set $w(\emptyset) = -\infty$; thus the set $\mathcal{W}$ consists of the element $-\infty$ plus the nonnegative real numbers. Bases have cardinality 0, 1, 2 or 3 (the 'interesting' cases are subsets with nonzero circumradius, where the basis cardinality is 2 or 3). A nontrivial step in verifying the axioms is to show that for any set, the minimum enclosing circle is unique, see [Wel91], [SW92]. Implementing the computational primitives is straightforward.

As our second example, we consider the *linear programming problem* in $\mathbb{R}^d$ in the following form: We are given a set H of closed halfspaces in $\mathbb{R}^d$, and the goal is to find the point x in the intersection of the halfspaces of H with the lexicographically smallest coordinate vector. Here the set $\mathcal{W}$ is formed by a smallest element $-\infty$, the set $\mathbb{R}^d$ ordered lexicographically by the coordinate vectors, and a largest element ∞. The value $w(G)$ of a set of constraints is defined as the lexicographically smallest point (vertex) of the intersection of all halfspaces of G. There are two exceptional cases: the value is ∞ if the intersection of the halfspaces of G is empty and it is $-\infty$ if this intersection is unbounded in the $-x_1$ direction or, more generally, if it does not have a lexicographically minimal point (e.g., if it is the halfspace $\{x_1 \geq 0\}$).

The verification of the axioms is easy. If the linear programming problem defined by H has a well defined optimum, then the combinatorial dimension of the problem is d; if the linear program is infeasible, the combinatorial dimension is usually $d+1$ (as $d+1$ halfspaces in general position are needed to witness the infeasibility).

Violation tests and basis changes for linear programming are implemented easily using Gauss elimination. The "Initial basis" assumption is more tricky, as it asks for finding a dually feasible solution if one exists. This can be circumvented by including the constraints "$x_i \geq -K$" for $i = 1, 2, \ldots, d$, with K standing for a very large number. These constraints can either be added to H explicitly and then they form an initial basis, or they can be included implicitly and then each set of constraints, including the empty one, defines a unique lexicographically minimal vertex, so there are no $-\infty$ values in this modification. In our discussion of linear programming in section 3, we assume that this latter variant has been chosen.

2.2 Nondegenerate LP-type problems

A nice feature of both the linear programming algorithms of Clarkson [Cla88] and of Sharir and Welzl [SW92] is that they do not require any special treatment of degenerate input configurations (such as many halfspace boundaries passing thru a single point). Probably for this reason, no notion of "general position" in the context of LP-type problems has been developed. For our algorithm below, some "nondegeneracy" is important, so we suggest a definition and make few observations concerning nondegenerate LP-type problems.

It is natural to define that an LP-type problem (H,w) is *nondegenerate* if $w(B) \neq w(B')$ for any two distinct bases B, B'. For a nondegenerate problem, we write $B(G)$ for the (unique) basis for a set G.

We must also define what does "removing degeneracies" mean. Let us say that an LP-type problem $(H,\bar{w})$ is a *refinement* of an LP-type problem (H,w), if for any $G, G' \subseteq H$, $w(G) < w(G')$ implies $\bar{w}(G) < \bar{w}(G')$. Clearly, if B is a basis in (H,w), it is also a basis in $(H,\bar{w})$, but not conversely. Removing degeneracies for a degenerate problem (H,w) means finding some its nondegenerate refinement.

When considering an LP-type problem (H,w) and its refinement $(H, \bar{w})$, we let the notation $V(G)$, resp. $B(G)$ refer to the set of violating constraints for G, resp. the basis for G in (H, w), and we use $\bar{V}(G)$, $\bar{B}(G)$ for violating constraints and basis in $(H, \bar{w})$.

There are two different types of degeneracies. To elucidate the difference, call two bases B, B' *equivalent* if they have identical sets of violating constraints, $V(B) = V(B')$. It is fairly easy to find a refinement of any LP-type problem where no two nonequivalent bases have the same value, as follows.

Define a linearly ordered set $\bar{\mathcal{W}} = \{-\infty\} \cup ((\mathcal{W} \setminus \{-\infty\}) \times 2^H)$, where $-\infty$ remains the smallest element, the pairs in the Cartesian product are ordered lexicographically and the ordering in the second coordinate is some arbitrary linear ordering of the set of all subsets of H. We define a new value function $\bar{w}$ by setting $\bar{w}(G) = (w(G), V(G))$; it is easy to check that this yields a refinement where no two nonequivalent bases have the same value[4].

Two equivalent bases must always have the same value. Any two equivalent but distinct bases thus violate the nondegeneracy condition, and this is an 'intrinsic' degeneracy. For instance, for linear programming, such degeneracies correspond to several bases defining the same vertex.

At present we are not aware of any universal and efficient method for removing this type of degeneracies. The following simple example shows that sometimes any nondegenerate refinement must have a larger dimension than the original (degenerate) problem.

Example 2.1 Let a, b, c, d be the vertices of a square in the plane in clockwise order, and for $G \subseteq H = \{a, b, c, d\}$ define the value $w(G)$ as the circumradius of G. This is an LP-type problem of dimension 2, as any nontrivial circumscribed circle is defined by a diametral pair. The sets $\{a, c\}$ and $\{b, d\}$ are two equivalent bases. Suppose there existed a nondegenerate refinement $(H, \bar{w})$ of this problem with no basis having more than 2 points. Then w.l.o.g. we may assume that $\bar{w}(\{a, c\}) < \bar{w}(\{b, d\})$. Set $G = \{a, b, c\}$, $F = \{a, c\}$ and $h = d$; then $\bar{w}(F) = \bar{w}(F \cup \{h\}) = \bar{w}(G) < \bar{w}(G \cup \{h\})$, which violates the locality axiom. Thus, any nondegenerate refinement has dimension at least 3.

For specific geometric problems, one can often assume that the corresponding geometric configurations are nondegenerate in a suitable sense (e.g., for linear programming, the boundary hyperplanes of the constraints are in general position and not parallel to any coordinate axis), using the techniques of *simulation of simplicity* for geometric problems, see e.g. [EM90], [Yap90], [EC92], [Sei94]. But, for instance, this is not enough for an infeasible linear programming problem. There may be many bases witnessing the infeasibility, and by the treatment of linear programming indicated above, all these bases receive the same value ∞. Another (related) serious problem is that if the feasible region of a degenerate linear programming problem consists of a single point (say), then some perturbations of the constraints may make the problem infeasible. Obtaining nondegenerate refinements in such cases may be a somewhat subtle matter; we discuss some possible approaches in section 3.

2.3 Bases of level k

Let (H, w) be an LP-type problem and $B \subseteq H$ be a basis. We define the *level* of B as $|V(B)|$, i.e. the number of constraints violating B.

We denote by $\mathcal{B}_k$ the set of all bases of level k, and by $\mathcal{B}_k^{(i)}$ the bases of level k and cardinality i. We use the notation $\mathcal{B}_{\leq k}$ for the set of all bases of level at most k.

In this section, we discuss bounds for the maximum possible size of $\mathcal{B}_{\leq k}$ and $\mathcal{B}_k$. In the context of linear programming (more exactly, for a feasible linear program), a basis of level k translates to a local minimum of the k-level in the arrangement of hyperplanes bounding the constraints. Mulmuley [Mul91] showed that the total number of such local minima of levels 1,2,...,k in dimension d is $O(k^d)$. His proof is a straightforward adaptation of the method of Clarkson and Shor [CS89]. Clarkson [Cla92] and independently Mulmuley [Mul93] proved that the number of local minima for level k alone is $O(k^{d-1})$. Both bounds are easily seen to be tight.

In the following theorem, we generalize these results for *nondegenerate* LP-type problems (the nondegeneracy is crucial; without it the bounds in the theorem below need not hold). The proof for level at most k goes through unchanged. For the exactly k level bound, we need an extra assumption, namely that no value $-\infty$ appears in the considered problem. Then we can imitate Clarkson's proof quite closely; the proof is omitted in this extended abstract. We do not know whether the statement remains true without this assumption.

Theorem 2.2 (i) *For any nondegenerate LP-type problem of dimension d, the number of bases of level at most k and of cardinality at most i does not exceed $e(k+1)^i$. In particular, $|\mathcal{B}_{\leq k}| = O((k+1)^d)$.*

(ii) *For any nondegenerate LP-type problem (H, w) of dimension d with $w(G) > -\infty$ for any $G \subseteq H$, we have $|\mathcal{B}_k| = O((k+1)^{d-1})$.*

[4] This modification of the weight function is not reflected in the computation of the algorithms which use only the primitive operations mentioned above, as none of these primitives are changed.

2.4 Finding the minimum k-level basis

In our algorithm, we need to assume that we deal with a nondegenerate LP-type problem. The following easy proposition (whose proof is omitted) shows that in order to find a minimal ($\leq k$)-level basis in a possibly degenerate LP-type problem (H, w), it suffices to find the minimum ($\leq k$)-level basis in any nondegenerate refinement of (H, w).

Proposition 2.3 *Let (H, w) be an LP-type problem, let $(H, \bar{w})$ be its nondegenerate refinement. Let $\bar{B}$ be the basis in $\bar{\mathcal{B}}_{\leq k}$ (the set of all bases of level at most k in $(H, \bar{w})$) with the minimum $\bar{w}$-value, let B be a basis for $\bar{B}$ in (H, w). Then $B \in \mathcal{B}_{\leq k}$ and $w(B)$ is minimum over $\mathcal{B}_{\leq k}$.*

For the rest of this section, let (H, w) be a nondegenerate LP-type problem of dimension d, and let $k \leq n$ be a given integer. Our goal is to find the basis with the smallest value among all bases of level at most k in (H, w). The algorithm consists of searching all bases of level at most k and selecting the one with minimum value. It it is easy to see that for a nondegenerate problem, the optimal basis has level exactly k, but our method requires searching also bases of smaller levels.

First we define a directed acyclic graph on the vertex set $\mathcal{B}_{\leq k}$. An edge goes from a basis $B \in \mathcal{B}_j$ to a basis $B' \in \mathcal{B}_{j+1}$ $(0 \leq j < k)$ if $V(B') = V(B) \cup \{b\}$ for some $b \in B$. We note that B' is the basis for $H \setminus V(B) \setminus \{b\}$. Therefore, given $B \in \mathcal{B}_j$, we can find all its neighbors in $\mathcal{B}_{j+1}$ by computing, for every $b \in B$, the basis B' for $H \setminus V(B) \setminus \{b\}$ and checking the condition $V(B') = V(B) \cup \{b\}$. This requires solving $|B| \leq d$ LP-type problems of the form (G, w) with $G \subseteq H$ plus simple auxiliary operations.

We need the following

Lemma 2.4 *Every basis of $\mathcal{B}_k$ can be reached from $B(H)$ by a directed path.*

Proof: It suffices to show that any basis $B' \in \mathcal{B}_{j+1}$ has a predecessor in $\mathcal{B}_j$.

Write $G = H \setminus V(B')$, then $B' = B(G)$. For every $h \in V(B') \neq \emptyset$, consider the value $w(G \cup \{h\})$, and let $h_0 \in V(B')$ be an element giving the smallest of these values. In fact, such h_0 is unique, as $w(G \cup \{h_0\}) = w(G \cup \{h\})$ for $h \neq h_0$ implies that $B(G \cup \{h_0\}) = B(G \cup \{h\}) \subseteq G$ by nondegeneracy, and this in turn means $w(G \cup \{h_0\}) = w(G)$, a contradiction with the assumption $h_0 \in V(B')$.

Let B be the basis for $G \cup \{h_0\}$. We claim that B is the desired predecessor of B'. We have $V(B) \cup \{h_0\} \subseteq V(B')$; we need to show equality. Suppose the inclusion is proper, then some $h \in V(B'), h \neq h_0$ does not violate B. Then, by locality, h does not violate $G \cup \{h_0\}$ either, so $w(G \cup \{h_0\}) = w(G \cup \{h, h_0\})$, but we have $w(G \cup \{h\}) > w(G \cup \{h_0\})$ by the choice of h_0, a contradiction with monotonicity. □

We are ready to finish the **proof of Theorem 1.2**. We start in the basis $B(H)$, and we search the above defined directed acyclic graph by some graph traversal algorithm, say by depth-first traversal. For any current node, we can find all of its at most d successors in $O(n)$ time, and by 2.2(i), we know that only $O(k^d)$ bases need to be searched. By the above lemma, we know that all bases of $\mathcal{B}_{\leq k}$ are reached. □

We conclude this section by a general scheme for an application of suitable dynamic data structures for implementing this algorithm (which more or less offers itself).

When traversing the graph defined above, we maintain two dynamic data structures, $\mathcal{O}$ and $\mathcal{V}$. For a current basis B, the data structure $\mathcal{O}$ stores the constraints of $H \setminus V(B)$, and it can quickly determine the basis for the currently stored set of constraints. The data structure $\mathcal{V}$ stores the constraints of $V(B)$, and it can quickly test whether all constraints of the currently stored set violate a basis given as a query.

For testing if a successor basis B' of B with $V(B') = V(B) \cup \{b\}$ for some $b \in B$ exists, we first delete b from $\mathcal{O}$ and then we query $\mathcal{O}$ for the basis B' for the current constraint set. Then we query the data structure $\mathcal{V}$ to check whether $V(B) \subseteq V(B')$. If yes, we insert b into $\mathcal{V}$, and we are ready to continue the search with B' as the current basis. When traversing the previously visited edge (B, B') backwards, we insert b to $\mathcal{O}$ and delete it from $\mathcal{V}$.

3 Geometric applications

In this section, we mainly consider Problem 1.3 (minimum vertex contained in all but at most k of given n halfspaces in $\mathbb{R}^d$). For a halfspace h, we let ∂h stand for its bounding hyperplane, and for a set G of halfspaces we write ∂G for $\{\partial h;\ h \in G\}$.

As outlined in section 2.1, the linear programming problem can be interpreted as an LP-type problem. One technical problem is that the input problem may be degenerate, while our algorithm requires a nondegenerate one. For a feasible linear program, degeneracy is caused by degeneracy in the arrangement of ∂H, while for an infeasible problem, any two distinct infeasible bases present a degeneracy, even if the hyperplanes of ∂H are in general position. For our algorithm, we need a nondegenerate refinement.

As we will see below, for a problem with a nonempty and full-dimensional feasible region, a nondegenerate refinement can be produced relatively easily using infinitesimal perturbations, while the situation gets more complicated in other cases. One potential problem is the following: if the feasible region is nonempty but has empty interior, an arbitrarily small perturbation of the input halfspaces may cause the problem to become infeasible, and in general we need not get a refinement.

At this moment, a general remark concerning simulation of simplicity is perhaps appropriate. When applying simulation of simplicity on a geometric problem, we replace it in effect by a different problem (although an "infinitesimally close" one), and an algorithm run on this different problem may sometimes yield a different answer than the correct one for the original problem (e.g., the original linear program is feasible while the perturbed one is not) . We can take two points of view: First, we may say that the input numbers for a "real world" problem are inaccurate anyway, and so the answer for the perturbed problem is equally appropriate as the one for the original problem. For example, we cannot really tell whether the feasible region is very thin or empty. From the second point of view, the input is given exactly and its degeneracies really exist, so if we use perturbation, we must be sure to recover the answer to the original problem correctly. By requiring that our algorithm is run on a nondegenerate refinement of the input problem, we are taking the second of the outlined positions (which may not always be appropriate, depending on the application). The first attitude would allow us to take an arbitrary infinitesimal perturbation, and would save us many complications.

The following proposition shows that a nondegenerate refinement can be constructed for a linear programming problem. Let $\mathcal{F}(H)$ denote the feasible region of the linear program defined by H.

Proposition 3.1 *Given a set H of halfspaces in $\mathbb{R}^d$, one can define a nondegenerate LP-type problem $(H, \bar{w})$, which is a refinement of the LP-type problem (H, w) defined above, has dimension d (resp $d+1$) if $\mathcal{F}(H) \neq \emptyset$ (resp. $\mathcal{F}(H) = \emptyset$), and for which the computational primitives can be implemented in constant time.*

Proof sketch: The refinement proceeds in several steps. First, we replace the problem of finding the lexicographically smallest feasible vertex by finding a vertex minimizing an objective function of the form $c \cdot x$, with $c = (1, \varepsilon_1, \varepsilon_1^2, \ldots, \varepsilon_1^{d-1})$. If $\varepsilon_1 > 0$ is chosen small enough, we get an isomorphic LP-type problem. Then we shift each of the constraint halfspaces by ε_2 outwards (keeping the bounding hyperplane parallel to the original one), with $O < \varepsilon_2 \ll \varepsilon_1$. This guarantees that whenever the feasible region is nonempty, it also has a nonempty interior. Then we perturb each bounding hyperplane by at most $\varepsilon_3 \ll \varepsilon_2$ (where distance of hyperplanes is defined as the distance of their dual points, say). This perturbation makes the values of all *feasible* bases distinct.

Finally we have to distinguish the values of the infeasible bases (this step is unnecessary for feasible problems). We consider an infeasible set G of halfspaces (after the final perturbation), and we divide them into upper and lower halfspaces. The value of G will be the smallest amount $t = t(G)$ by which we must shift the intersection of all lower halfspaces upwards so that it touches the intersection of all upper halfspaces. It turns out that this gives the desired nondegenerate refinement of dimension $d+1$; we omit the details. In the algorithm, we do not choose specific numbers for $\varepsilon_1, \varepsilon_2, \varepsilon_3$, rather we treat them as indeterminates. □

By proving Proposition 3.1, we have shown that the general algorithm from Theorem 1.2 is applicable to Problem 1.3. Further we consider efficient implementation of the dynamic data structures $\mathcal{O}$ and $\mathcal{V}$ mentioned in the end of section 2.4 for the linear programming problem (actually for its nondegenerate refinement constructed in the proof of Proposition 3.1).

We begin with the feasible case. Here the data structure $\mathcal{V}$ stores a set of halfspaces which all contain a known point (a fixed point in $\mathcal{F}(H)$), and the query boils down to checking whether a given point lies outside of the union of the current set of halfspaces. For this task, algorithms are known with the following performance [AM91] (we only quote the results relevant to our application): In dimension 3, $O(n^\varepsilon)$ query time can be achieved with $O(n \log n)$ preprocessing time and $O(\log^2 n)$ amortized update time. In dimension $d \geq 4$, one gets the following tradeoff: for a parameter m in range $[n, n^{\lfloor d/2 \rfloor}]$, query time $O(n^{1+\varepsilon}/m^{1/\lfloor d/2 \rfloor})$ is obtained with $O(m^{1+\varepsilon})$ space and preprocessing time and with $O(m^{1+\varepsilon}/n)$ amortized update time (in fact, for $m = n$ an $O(n \log n)$ preprocessing suffices). We must apply the appropriate perturbations on the input halfspaces, but this is a simulation of simplicity technique of a particular type and it only slows down the computation by a constant factor.

The data structure $\mathcal{O}$ should return a basis for the current set of halfspaces (and we are guaranteed that the halfspaces have a nonempty intersection). Results of [Mat93a] show that the same performance can be achieved as the one above for $\mathcal{V}$ (the query time grows by a polylogarithmic factor, which formally disappears in the n^ε term).

In dimension 3, we perform $O(k^3)$ queries and updates in both data structures, which leads to the claimed $O(n \log n + k^3 n^\varepsilon)$ complexity. For dimension 4, we choose a suitable tradeoff between the total update and query time and the preprocessing time of the data structures, as follows: For $k < n^{1/8}$, we let $m = n$, for $k^4 \leq n < k^8$ we let $m = k^{8/3} n^{2/3}$ and for larger k we choose $m = n^{4/3}$. This yields the formula stated in the theorem. Tradeoffs can be computed also for higher dimensions, although with less significant gains in efficiency. This establishes part (i) of Theorem 1.4.

In the infeasible case, the data structure $\mathcal{V}$ stores the upper halfspaces and lower halfspaces separately. Testing for a feasible basis is as before. For an infeasible basis B, we first compute the value $t(B)$ as in the proof of Proposition 2.3 and the point $\tau = \tau(B)$ of the intersection of all upper halfspaces of B where the intersection of all lower halfspaces of B first touches when shifted upwards, and then we test if τ lies outside all upper halfspaces and $\tau - (t, 0, \ldots, 0)$ lies outside all lower halfspaces.

The data structure $\mathcal{O}$ also stores upper and lower halfspaces separately. If the currently stored set is feasible, the algorithm of [Mat93a] returns the basis. If infeasibility is reported, we use parametric search to find the value of t. As a generic algorithm, we use the algorithm of [Mat93a] for testing feasibility of the set of the upper halfspaces plus the lower halfspaces shifted by a generic value of t. A more detailed exposition would require explaining the feasibility testing algorithm and we omit it, as the details are easy to fill in, assuming a familiarity with [Mat93a]. The application of parametric search on top of the feasibility testing algorithm only increases the query time by polylogarithmic factors. The overall performance of the resulting dynamic data structures $\mathcal{V}$ and $\mathcal{O}$ is thus the same as in the feasible case above; this gives the result for dimension 3 in Theorem 1.4(ii).

For the 2-dimensional case, the parametric search machinery is unnecessary, and we may directly use a relatively simple dynamic data structure for maintaining convex hulls in the plane due to Overmars an van Leeuwen [OvL81]. To build $\mathcal{O}$, we use this data structure in a dual form. One part represents the intersection U of the upper halfplanes, another part the intersection L of the lower halfplanes. After $O(n \log n)$ preprocessing, halfplanes can be inserted and deleted in $O(\log^2 n)$ time, and the data structure provides a representation of the convex chains forming the boundaries of L and U (the chains are stored in balanced binary trees).

If the current problem is feasible, the optimal vertex is either the extreme vertex of U, or the extreme vertex of L, or one of the two intersections of the boundaries of U and of L. All these vertices can be found and examined in $O(\log^2 n)$ time. For an infeasible problem, the first contact of U and L when translating L upwards occurs either at a vertex of L or at a vertex of U. For a given vertex v of L, we can determine the point where it hits the boundary of U in $O(\log n)$ time. From the local situation at that point, we can detect whether the first contact of L and U occurs to the left of v or to the right of v. Hence we can determine the first contact, the corresponding t value and the basis in $O(\log^2 n)$ time by binary search in the lower convex chain. The data structure $\mathcal{V}$ is also implemented using the Overmars and van Leeuwen data structure, with $O(\log^2 n)$ time per update and $O(\log n)$ time per query. Altogether we get the $O(n \log n + k^3 \log^2 n)$ running time. This finishes the proof of Theorem 1.4. □

Proof of Theorem 1.1: For the smallest enclosing circles problem discussed in the introduction, the situation with a nondegenerate refinement is considerably easier than the one for linear programming (this is because the value function depends continuously on the point set). It suffices to take any sufficiently small perturbation of the input points such that no 4 points are cocircular and no circle determined by 2 points passes through another point.

To build the required dynamic data structures, we use the well-known lifting transformation. Namely, we map each point $p = (u, v)$ in our point set to the plane $\pi(p) = \{(x, y, z) \in \mathbb{R}^3;\ z = u^2 + v^2 - 2ux - 2vy\}$, and we map a circle C with center (a, b) and radius r to the point $x(C) = (a, b, r^2 - a^2 + b^2) \in \mathbb{R}^3$. Then p lies inside C iff $x(C)$ lies above $\pi(p)$.

This allows us to implement the data structure $\mathcal{V}$ using the dynamic data structure of [AM91] mentioned in the previous section. For the data structure $\mathcal{O}$, we must work somewhat more.

For a point set P in the plane, let $A(P)$ denote the cell above all the planes $\pi(p)$ with $p \in P$. Finding the smallest circle enclosing P translates to minimizing the function $\varphi(x, y, z) := x^2 + y^2 - z$ over the region $A(P)$; the minimum value is the square of the radius of the smallest enclosing circle, and the x and y coordinates of the point realizing the minimum determine the center of the circle.

The procedure is quite analogous to the one from [Mat93a] for minimizing a linear function over a convex polyhedron (which is given implicitly by a certain algorithm for answering membership queries), only that here we minimize a convex function. In our case, the membership algorithm is the query answering algorithm for the dynamic data structure of [AM91]. A reader familiar with the procedure of [Mat93a] may check that the only point where the difference between a linear function and a convex one appears is in the following situation: We have a j-flat f and a $(j-1)$-flat $f' \subset f$. We know that the minimum of φ over $f' \cap A(P)$ is attained at some point $z \in f'$. From the local geometry near this point we want to determine on which side of f' lies the minimum of φ within $f \cap A(P)$. In case of minimizing a linear function, the minimum point z is determined by $j-1$ of the planes bounding $A(P)$, and one can easily determine on which side of f' the minimized function may grow. In our case of a (quadratic) convex minimized function, the minimum may be the global minimum, or it may be determined by any number of planes up to $j-1$, but still one may determine locally on which side of f' does the minimum within f lie. With this subroutine available, we may use the procedure of [Mat93a], and we obtain a minimization algorithm with $O(n^\varepsilon)$ query time. Generalization to an arbitrary dimension is also possible (the efficiency depends on the efficiency of the algorithm for the membership queries).

In this way, also the data structure $\mathcal{O}$ can be implemented with $O(n^\varepsilon)$ query time, and this gives the bound in Theorem 1.1. □

There are various other optimization problems of geometric flavor fitting into the LP-type framework, see [MSW92], [Ame93]. Here are few examples: Finding the smallest ball, resp. the smallest volume ellipsoid enclosing a given point set in $\mathbb{R}^d$; finding the largest volume ellipsoid inscribed into the intersection of given halfspaces in $\mathbb{R}^d$; finding the distance of two convex polyhedra given by vertices, resp. by facet hyperplanes in $\mathbb{R}^d$; finding a line transversal for given convex polygons in the plane.

If the dimension is fixed, one can apply our general algorithm to the respective derived problems with k violated constraints, provided that the nondegeneracy issue can be handled. In many of the problems, simulation of simplicity alone should suffice; it seems that linear programming is complicated in this respect because feasible solutions need not exist. The applicability of dynamic data structures to speed up the computations must be checked individually. In general, the improvements will probably be significant only if the dimension is really small.

4 Discussion

It would be interesting to find more about nondegenerate refinements of LP-type problems. We have shown that the dimension must sometimes grow at least by 1; a natural question is how large growth is necessary and sufficient in the worst case; is there any bound only depending on the dimension?

From a practical point of view, it would be very desirable to have some more direct scheme for making the geometric LP-type problems, linear programming in particular, nondegenerate that the rather cumbersome approach via geometric perturbations we used. Alternatively one might find an algorithm which can handle nondegeneracy directly.

In Theorem 2.2(ii), we saw that the number of bases of level exactly k is $O(k^{d-1})$ in a d-dimensional LP-type problem with no $-\infty$ values. Two questions arise naturally: First, is the claim still true if we allow $-\infty$ values? And second, can one use this result algorithmically for finding the smallest basis of level k, that is, can one efficiently avoid searching all bases of level *at most* k, whose number may be of the order k^d? In particular, for the 2-dimensional infeasible linear programming, we are actually interested only in 2-element bases (as all infeasible bases have the same value ∞ in the original problem), and we know that there are only $O(k^2)$ of these. Still, the current method may search k^3 bases, most of them infeasible ones; could this be avoided?

Acknowledgment. I would like to thank Emo Welzl for bringing the problem to my attention, and Pankaj K. Agarwal and David Eppstein for useful discussions.

References

[ACE+91] B. Aronov, B. Chazelle, H. Edelsbrunner, L. J. Guibas, M. Sharir, and R. Wenger. Points and triangles in the plane and halving planes in space. *Discrete Comput. Geom.*, 6:435–442, 1991.

[AdBMS94] P. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. In *Proc. 10th Annual ACM Symposium on Comput. Geom.*, 1994. To appear.

[AIKS91] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *J. Algorithms*, 12:38–56, 1991.

[AM91] P. K. Agarwal and J. Matoušek. Dynamic halfspace range reporting and its applications. Tech. Report CS-1991-43, Duke University, 1991. Extended abstract, including also results of D. Eppstein: *Proc. 33. IEEE Symposium on Foundations of Computer Science* (1992), pages 51–60.

[Ame93] N. Amenta. Helly theorems and generalized linear programming. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 63–72, 1993. Also to appear in *Discrete & Computational Geometry.*

[Cla87] K. L. Clarkson. New applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 2:195–222, 1987.

[Cla88] K. L. Clarkson. A Las Vegas algorithm for linear programming when the dimension is small. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 452–456, 1988.

[Cla92] K. L. Clarkson. A bound on local minima of arrangements that implies the upper bound theorem. Manuscript, 1992.

[CM93] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 281–290, 1993.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[CSY87] R. Cole, M. Sharir, and C. K. Yap. On k-hulls and related problems. *SIAM J. Comput.*, 16:61–77, 1987.

[DE93] T. Dey and H. Edelsbrunner. Counting triangle crossings and halving planes. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 270–273, 1993.

[DLSS93] A. Datta, H.-P. Lenhof, C. Schwarz, and M. Smid. Static and dynamic algorithms for k-point clustering problems. In *Proc. 3rd Workshop Algorithms Data Struct.*, Lecture Notes in Computer Science, 1993.

[EC92] I. Emiris and J. Canny. An efficient approach to removing geometric degeneracies. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 74–82, 1992.

[EE93] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 64–73, 1993.

[ELS93] A. Efrat, M. Lindenbaum, and M. Sharir. Finding maximally consistent sets of halfspaces. In *Proc. 5th Canadian Conference on Computational Geometry*, 1993.

[EM90] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, 9:66–104, 1990.

[ERvK93] H. Everett, J.-M. Robert, and M. van Kreveld. An optimal algorithm for the ($\leq k$)-levels with applications to separation and transversal. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 38–46, 1993.

[ES93] J. Erickson and R. Seidel. Better lower bounds for detecting affine and spherical degeneracies. In *Proc. 34th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS 93)*, 1993.

[ESZ93] A. Efrat, M. Sharir, and A. Ziv. Computing the smallest k-enclosing circle and related problems. In *Workshop on Algorithms and Data Structures (WADS)*, 1993.

[Gär92] B. Gärtner. A subexponential algorithm for abstract optimization problems. In *Proc. 33rd IEEE Sympos. Found. Comput. Sci.*, 1992.

[GO93] A. Gajentaan and M. Overmars. n^2-hard problems in computational geometry. Tech. Report RUU-CS-93-15, Utrecht University, 1993.

[Kal92] G. Kalai. A subexponential randomized simplex algorithm. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 475–482, 1992.

[Mat93a] J. Matoušek. Linear optimization queries. *Journal of Algorithms*, 14:432–448, 1993. The results combined with results of O. Schwarzkopf also appear in *Proc. 8th ACM Symposium on Computational Geometry*, 1992, pages 16–25.

[Mat93b] J. Matoušek. On enclosing k points by a circle. Manuscript, 1993.

[Meg83] N. Megiddo. The weighted Euclidean 1-center problem. *Math. Oper. Res.*, 8(4):498–504, 1983.

[MSW92] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 1–8, 1992. Also to appear in *Algorithmica.*

[Mul91] K. Mulmuley. On levels in arrangements and Voronoi diagrams. *Discrete Comput. Geom.*, 6:307–338, 1991.

[Mul93] K. Mulmuley. A generalization of Dehn-Sommerville relations to levels in arrangements. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 1993.

[OvL81] M. H. Overmars and J. van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981.

[Sei94] R. Seidel. The nature and meaning of perturbations in geometric computing. Manuscript, 1994.

[SW92] M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 1992 Symposium on Theoretical Aspects of Computer Science (Lecture Notes in Computer Science)*, volume 577, pages 569–579. Springer-Verlag, 1992.

[Wel91] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *LNCS 555 (New Results and New Trends in Computer Science)*, pages 359–370. Springer-Verlag, 1991.

[Yap90] C. K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *J. Comput. Syst. Sci.*, 40:2–18, 1990.

Efficient Piecewise-Linear Function Approximation Using the Uniform Metric

(Preliminary Version)

Michael T. Goodrich*

Dept. of Computer Science
Johns Hopkins Univ.
Balitmore, MD 21218
Email: goodrich@cs.jhu.edu

Abstract

We give an $O(n \log n)$-time method for finding a best k-link piecewise-linear function approximating an n-point planar data set using the well-known uniform metric to measure the error, $\epsilon \geq 0$, of the approximation. Our method is based upon new characterizations of such functions, which we exploit to design an efficient algorithm using a plane sweep in "ϵ space" followed by several applications of the parametric searching technique. The previous best running time for this problem was $O(n^2)$.

1 Introduction

Given a set $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, the problem of approximating S by a function is classic in applied mathematics, and it finds applications in a number of computational problems. The general goals in this area of research are to find a function F belonging to a class of functions $\mathcal{F}$ such that each $F \in \mathcal{F}$ is simple to describe, represent, and compute and such that the chosen F approximates S well. For example, one may desire that $\mathcal{F}$ be the class of linear or piecewise-linear functions, and, for any particular $F \in \mathcal{F}$, that the measure of the error be the well-known *uniform* metric:

$$||S - F||_\infty = \max_{i \in \{1,2,\ldots,n\}} |y_i - F(x_i)|,$$

which is also known as the l_∞ or *Chebychev* measure of error [16, 18, 30]. The goal, then, is to determine the value of

$$\epsilon^* = \min_{F \in \mathcal{F}} ||S - F||_\infty,$$

and find an $F \in \mathcal{F}$ achieving this error bound.

The version of this problem we address in this paper is to find a function $F \in \mathcal{F}$ that minimizes the uniform error term with respect to S where $\mathcal{F}$ is the class of k-link piecewise-linear functions, for some given $k \in \{1, 2, \ldots, n-1\}$. Using terminology from the approximation theory literature (e.g., see [9, 16, 18, 19]), this is equivalent to the problem of finding a best $(k+1)$-knot degree-1 spline approximating S under the l_∞ norm. Of course, the case $k = n - 1$ is trivial, and there is a simple reduction of the case $k = 1$ to 3-dimensional linear programming, which can be solved in $O(n)$ time [12, 20, 40, 42, 43, 53]. Thus, the interesting cases are for $1 < k < n - 1$. We show how to solve this problem for any such k in $O(n \log n)$ time.

The motivation for this problem is that one may have limited resources with which to describe the set S, but one wishes the best approximation possible within the given resource bounds. This can also be viewed as a data compression problem.

1.1 Previous work

The problem we address is a special case of a whole class of problems in approximation theory where one wishes to fit a set of data using a spline function under some metric. Thus, the interested reader is referred to texts discussing approximation theory, such as those by Bellman and Roth [9], Conte and de Boor [16], Davis [18], and Dierckx [19], for a general treatment of such problems. Research in this literature is primarily interested in minimizing the number of knots in a spline under the the least squares metric, e.g., Jupp [36] gives a numerical approach to this problem. For the specific problem we address here, Bellman and Roth [8] describe a dynamic-programming approach based upon using a uniform grid to determine possible placements of link endpoints (which they call *knots*). Their method is not guaranteed to find a best k-link approximation, however.

Hakimi and Schmeichel [30] show that such a best approximation can be found in $O(n^2 \log n)$ time, and this is the first method we know of that is guaranteed

*Partially supported by the NSF and DARPA under Grant CCR-8908092, and by the NSF under Grants IRI-9116843 and CCR-9300079.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

to find a best approximation. Their algorithm is based upon a clever lemma that shows that one can limit the number of "critical" ϵ values that are candidates for ϵ^* to be $O(n^2)$. They also show that one can test if any such ϵ value is equal to ϵ^* in $O(n)$ time, which implies that, once enumerated and sorted, one can perform a "binary search" among these critical values to find ϵ^*. Of course, enumerating these critical ϵ's requires $\Omega(n^2)$ time. Indeed, a straightforward application of the lemma by Hakimi and Schmeichel would require $O(n^3)$ time to enumerate them. They reduce the time to $O(n^2 \log n)$ using the powerful *plane sweeping* technique (e.g., see [51]), which involves "sweeping" the plane with a line L while maintaining appropriate data structures for the points L encounters along the way. More recently, Wang *et al.* [58] show how to use an even more clever plane sweep procedure to find a best k-link approximation under the uniform metric in $O(n^2)$ time.

1.2 Related work

With the exception of the papers by Hakimi and Schmeichel [30] and Wang *et al.* [58], related work in the computational geometry literature has been directed at what can be viewed as the "inverse problem," which is also addressed in the paper by Hakimi and Schmeichel [30]. In this problem one is given an error measure $\epsilon \geq 0$ and asked to find a minimum-link polygonal path (which may or may not be required to be a function) that has distance at most ϵ from all the objects in S (which need not just be points), under some reasonable distance metric. As mentioned earlier, for the case when S is a set of points and the error measure is the uniform metric, then Hakimi and Schmeichel show that this problem can be solved in $O(n)$ time. Their method can be viewed as an extension of the linear-time method of Suri [54], which computes a minimum-link path inside a simple polygon, to the problem of finding a minimum-link monotone polygonal chain that "stabs" a given set of line segments. Hershberger and Snoeyink [32] show how to further generalize this method to find in $O(n)$ time a minimum-link path of a particular homotopy type in a non-simple polygon, and Guibas, Hershberger, Mitchell, and Snoeyink [29] show how to generalize this method even further to find in $O(n)$ time a minimum-link stabber for any given set of disjoint convex objects that must be stabbed in some given order (not necessarily just by increasing x-coordinates). Robert and Toussaint [52] study the problem of finding a line L that minimizes a weighted minmax error measure to a set of convex polygons in $O(n^2 \log n)$ time.

There has also been a considerable amount of work on finding a minimum-link approximation to a polygonal curve, subject to some error tolerance. The problem of fitting a minimum-link convex polygon nested between two given polygons was studied by Aggarwal *et al.* [5], who give an $O(n \log n)$ time solution to this problem. In addition, Imai and Iri [34, 35] give an $O(n)$ time method for finding the minimum-link function approximating a given monotone chain. Their method is very similar to an $O(n)$ time method independently discovered by Suri [54, 55] for solving the more-general problem of finding a minimum link path joining two points inside a simple polygon. There has also been some work on approximations that are required to use a subset of the endpoints of the given polygonal chain. For example, using an approach of Imai and Iri [33, 35], Toussaint [57] and Melkman and O'Rourke [44] give several $O(n^2 \log n)$ time methods under various metrics.

There hasn't been much work on three-dimensional version of these approximation problems with guaranteed performance bounds, however, although the recent work by Mitchell and Suri [49] on a special case of the 3-dimensional function approximation problem is a notable exception.

There is also a rich literature that studies minimum-link distance as a metric in its own right (e.g., see [6, 23, 24, 37, 39, 47, 48, 54, 55]).

1.3 Our results

As mentioned above, we give an $O(n \log n)$ time algorithm for finding a best k-link piecewise-linear function approximating a set S of n points in the plane under the uniform metric. Our method is based upon new geometric insights that allow us to apply a novel plane sweep in "ϵ space" to enumerate a set of $O(n)$ critical ϵ values, which we then search in a binary search fashion. This allows us to restrict the range of ϵ values containing ϵ^* to be an interval $[\epsilon_1, \epsilon_2]$, but it does not necessarily give us $\epsilon_1 = \epsilon_2 = \epsilon^*$. To achieve this latter result we give additional geometric characterizations of a best k-link approximation that allow us to follow this preprocessing step by several applications of pipelined versions of the well-known *parametric searching* technique (e.g., see [2, 3, 4, 11, 13, 14, 15, 41]).

Admittedly, the use of this technique typically makes an algorithm rather impractical to implement. But we show that this is not true in our case, for we can design a relatively simple version of our algorithm that uses only the most simple versions of parametric searching (which can be made even more practically efficient via randomization).

In the section that follows we give some properties of a best k-link approximation and in Section 3 we show how to exploit these properties to restrict the range of candidate ϵ values. We show how to then complete the

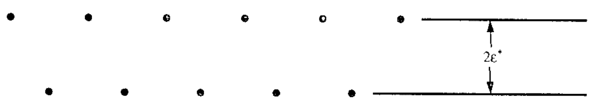


Figure 1: An example set S such that $S(\epsilon^*)$ has a 1-link ordered stabber, but any ordered stabber of $S(\hat{\epsilon})$ requires 10 links if $\hat{\epsilon} < \epsilon^*$.

construction in Section 4 by relying on additional geometric properties of a best k-link approximation, which we show can be exploited in a series of applications of parametric searching. Finally, we show how to simplify our implementation in Section 5.

2 Some Properties of a Best k-Link Approximation

Let $S = (p_1, p_2, \ldots, p_n)$ be a left-to-right ordered listing of the points in S and let $\epsilon \geq 0$ be given. So as to formally define our approximation problem and articulate some of its important properties, let us introduce a bit of additional notation. For each point $p_i = (x_i, y_i)$ in S define $u_i = (x_i, y_i + \epsilon)$ and $g_i = (x_i, y_i - \epsilon)$, and let $S(\epsilon)$ denote the ordered set of vertical segments $(\overline{u_1 g_1}, \overline{u_2 g_2}, \ldots, \overline{u_n g_n})$. Thus, if we view points as degenerate segments, then $S = S(0)$. For any ordered set of disjoint geometric objects A a polygonal chain C is an *ordered stabber* if a traversal of C intersects the objects of A in the given order [29]. Finally, define $F(\epsilon)$ to be a minimum-link ordered stabber of $S(\epsilon)$.

The formal problem we address in this paper, then, is to find ϵ^*, the smallest $\epsilon \geq 0$ such that $F(\epsilon)$ has at most k links. Formulating the problem in this way allows us to deal with "degenerate" inputs, such as the one illustrated in Figure 1, where $S(\epsilon^*)$ may allow an ordered stabber with $k' < k$ links, but a minimum-link stabber of $S(\hat{\epsilon})$ may require $\hat{k} > k$ links for any $\hat{\epsilon} < \epsilon^*$. Thus, a best k-link approximation $F = F(\epsilon^*)$ may, in fact, have fewer than k links because of degeneracies. Of course, one can always introduce "dummy" vertices along F to force its link-count to be exactly k in such a case.

2.1 A Canonical Form for Best k-Link Approximations

Let us connect consecutive g_i's and u_i's so as to form two "parallel" monotone chains $U(\epsilon)$ and $G(\epsilon)$, with $U(\epsilon)$ being the upper chain, i.e., let us create edges $\overline{g_i g_{i+1}}$ defining $G(\epsilon)$ and $\overline{u_i, u_{i+1}}$ defining $U(\epsilon)$ for $i \in \{1, 2, \ldots, n-1\}$. One might be tempted to think that a best k-link function F approximating S can be constrained to lie between $U(\epsilon^*)$ and $G(\epsilon^*)$, but this is not the case[1] (as shown in Figure 2). This is actually a good thing, for otherwise we would run into some robustness difficulties, for we would have to use a method for finding a minimum-link path in a simple polygon as a subroutine, and, as Snoeyink observers[2], the bit complexity for finding such a path can be signficantly larger than the bit complexity for representing the vertices of the input polygon. This is no problem for our method, however, for we will be using methods for finding minimum-link stabbers as subroutines in our algorithm, and these methods do not suffer from this bit complexity blow-up difficulty.

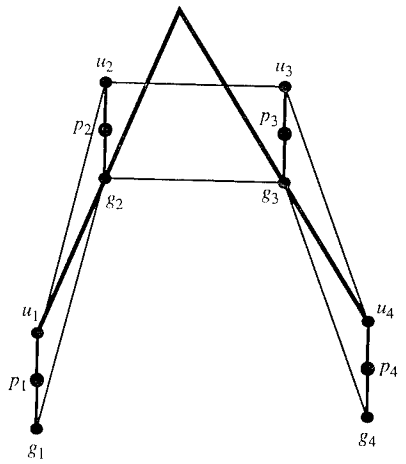


Figure 2: An instance where a minimum-link stabber of $S(\epsilon)$ is not confined to lie between $U(\epsilon)$ and $G(\epsilon)$.

To describe why we can use minimum-link stabber methods as subroutines, we must show how to restrict F to a certain canonical form. For a given ϵ, let $\pi_u(\epsilon)$ be the shortest path from u_1 to u_n that does not go above $U(\epsilon)$ and does not cross $G(\epsilon)$. Similarly, let $\pi_g(\epsilon)$ be the shortest path from g_1 to g_n that does not go below $G(\epsilon)$ and does not cross $U(\epsilon)$. Such paths were introduced by Lee and Preparata [38] and are often referred to as *geodesic* paths [1, 7, 10, 25, 28, 45, 46, 56], and the union of two such paths is often called an *hourglass* [22, 26]. Let us therefore use $H(\epsilon)$ to denote this hourglass $\pi_u(\epsilon) \cup \pi_g(\epsilon)$. We say that an edge of $H(\epsilon)$ is an *inflection* edge if one of its endpoints lies on $U(\epsilon)$ while its other endpoint lies on $G(\epsilon)$. Let $I(\epsilon)$ denote the set of all such inflection edges. (See Figure 3.)

We say that two consecutive links $\overline{pq}$ and $\overline{qr}$ in F have a *zig* turn type if r is above the ray $\overrightarrow{pq}$ (i.e., $\overline{pq}$ and $\overline{qr}$ form a "left turn"). Similarly, two consecutive

[1] We are indebted to Jack Snoeyink (personal communication) for pointing this example out to us.

[2] Again, by a personal communication.

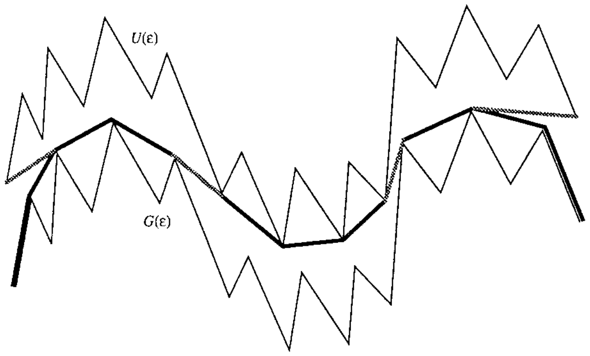


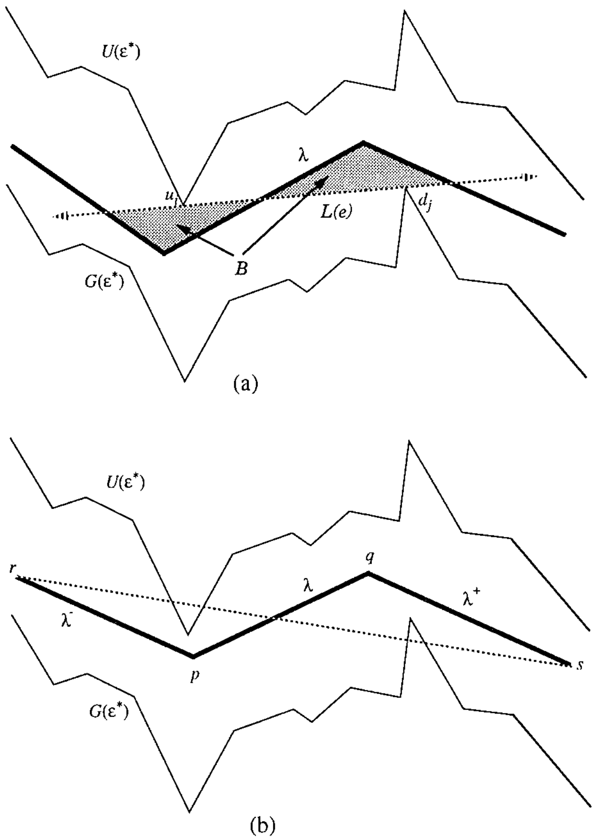
Figure 3: An example hourglass. The inflection edges are shown shadded.

links $\overline{pq}$ and $\overline{qr}$ in F a *zag* turn type if r is below the ray $\overrightarrow{pq}$. This allows us to characterize each link in F, other than the first and last links, by the turn types they form with their predecessor and successor links. For example, a zig-zag link forms a left turn with its predecessor and a right turn with its successor. The next lemma establishes an important relationship between such links and inflection edges in $I(\epsilon)$.

Lemma 2.1: *There is a best k-link function F approximating S such that*

1. *each $e \in I(\epsilon^*)$ is contained by the first or last link of F or by a zig-zag or zag-zig link of F, and*
2. *the first and last link of F, as well as each zig-zag and zag-zig link of F, contains an $e \in I(\epsilon^*)$.*

Proof: (1:) Suppose $e \in I(\epsilon^*)$, i.e., e is an inflection edge of $H(\epsilon^*)$. Also suppose, for the sake of contradiction, that e is contained in no link of the appropriate type in any best k-link approximation F. We will follow a proof technique of Ghosh [23], which involves performing local perturbations of a candidate stabber, to derive a contradiction. Since e is an inflection edge, it connects a u_i to a g_j; hence, F must intersect e along some link λ, for F cannot go above any u_i nor below any g_j. Let us assume for the time being that λ is neither the first nor last link of F, and let λ^- denote λ's predecessor and let λ^+ denote λ's successor. Also, let $L(e)$ denote the line containing e and let B denote the set of all points on some segment from a point p on λ^-, λ, or λ^+ to p's nearest neighbor on $L(e)$, i.e., the points "between" $L(e)$ and λ^-, λ, and λ^+. B can contain no points of $U(\epsilon^*)$ nor $G(\epsilon^*)$, for if this were not the case, then e would not be an inflection edge (e.g., see Figure 4a). Thus, we can "move" the common endpoint of λ^- and λ to be on $L(e)$, and the common endpoint of λ and λ^+ to be on $L(e)$, keeping the rest of F fixed, and we will keep F as a k-link approximation to S. To



(a)

(b)

Figure 4: Example zig-zag edges. In (a) we illustrate why an inflection edge is contained in a zig-zag link, and in (b) we illustrate why a zig-zag link contains an inflection edge.

establish that λ must be a zig-zag or zag-zig link, note that the first place where the ray $\overrightarrow{u_i g_j}$ crosses $G(\epsilon^*)$ cannot be before the first place it crosses $U(\epsilon^*)$, and, likewise, the first place where the ray $\overrightarrow{g_j u_i}$ crosses $U(\epsilon^*)$ cannot be before the first place it crosses $G(\epsilon^*)$ (e.g., see Figure 4a). If this were not so, $\overline{u_i g_j}$ would not be an inflection edge of $H(\epsilon^*)$. Thus, either λ is the first or last link of F (which occurs if one of the rays $\overrightarrow{u_i g_j}$ or $\overrightarrow{g_j u_i}$ crosses neither $U(\epsilon^*)$ nor $G(\epsilon^*)$) or λ is a zig-zag or zag-zig link (since F is a minimum-link approximation to $S(\epsilon^*)$). Similar (actually simpler) arguments hold for the cases when λ^- or λ^+ do not exist, and are left to the reader. Therefore, there is a best k-link approximation to S that contains each edge in S, and each such edge is contained in the first or last link of F or in a zig-zag or zag-zig link.

(2:) For the second part of the lemma, let m be the minimum number of zig-zag, zag-zig, first, and last links that do not contain any edge in $I(\epsilon^*)$, taken over all best k-link approximations to S satisfying part one of the lemma (which we have just shown to be true). In addition, let $\lambda = \overline{pq}$ be one of these m links. Let us assume for the time being that λ is zig-zag link in F. Let $\lambda^- = \overline{rp}$ denote the predecessor of λ in F and let $\lambda^+ = \overline{qs}$ denote the successor of λ in F. Since F is a minimum-link path, the line $\overline{rs}$ must intersect both $U(\epsilon^*)$ and $G(\epsilon^*)$(e.g., see Figure 4b). But this implies that F crosses an inflection edge, which is a contradiction; hence, we establish the second part of the lemma for this case. The proofs for the other cases are similar; hence, this establishes the lemma. □

Having established an important property of some of the links in a best k-link function approximation to S, let us now turn to the problem of enumerating these edges, and in the process we will also restrict the range of ϵ's that allow a k-link approximation.

3 Finding the Inflection Edges

We say that an ϵ is *geodesic-critical* if $H(\epsilon)$ has l edges, but $H(\hat{\epsilon})$ has fewer than l edges for $\hat{\epsilon} > \epsilon$. Our method for finding all the inflection edges is to determine an interval $[\epsilon_1, \epsilon_2]$ that contains ϵ^* and is such that $H(\epsilon_1)$ is combinatorially equivalent to $H(\epsilon_2)$. This will allow us to determine all inflection edges that F must contain. Our procedure is conceptually quite simple. First we enumerate all $O(n)$ geodesic-critical ϵ values, and then we perform a binary search among these values to determine the interval $[\epsilon_1, \epsilon_2]$ containing ϵ^*.

3.1 Enumerating all geodesic-critical ϵ values

Our method for enumerating all geodesic-critical ϵ values is based upon a sweep through "ϵ space." We maintain the hourglass $H(\epsilon) = \pi_u(\epsilon) \cup \pi_g(\epsilon)$ while taking ϵ from 0 to $+\infty$, stopping at each geodesic-critical ϵ value along the way. To simplify the discussion, however, let us concentrate on the problem of maintaining $\pi_u(\epsilon)$, so that we restrict our notion of geodesic-critical ϵ's to those that change $\pi_u(\epsilon)$; the method for maintaining $\pi_g(\epsilon)$ is similar. Initially, for $\epsilon = 0$, $\pi_u(\epsilon)$ is the chain $U(\epsilon) = G(\epsilon)$; hence, it consists of $n-1$ edges. If we then increase ϵ by an "infinitesimal" amount we find that some of the vertices of $\pi_u(\epsilon)$ lie on $U(\epsilon)$ while others lie on $G(\epsilon)$. For any vertex p on $\pi_u(\epsilon)$, if one of p's adjacent vertices on $\pi_u(\epsilon)$ lies on the chain opposite from the chain that p lies on, then we say that p is a *pinch* vertex. For each pinch vertex p on $\pi_u(\epsilon)$, let q and r denote p's adjacent vertices on $\pi_u(\epsilon)$, and compute the $\hat{\epsilon} > \epsilon$ value at which p would cease to be a pinch vertex if we were to restrict $U(\hat{\epsilon})$ and $G(\hat{\epsilon})$ to that portion of the plane bounded by the lines $x = x(q)$ and $x = x(r)$ inclusive, where $x(t)$ denotes the x-coordinate of a point t. Call this $\hat{\epsilon}$ value *locally-critical* for p, and let E be the set of all $\hat{\epsilon}$'s that are locally-critical for pinch vertices on $\pi_u(\epsilon)$.

Lemma 3.1: *The smallest $\hat{\epsilon}$ in E is the smallest geodesically-critical value bigger than ϵ.*

Proof: Let ϵ' be the smallest geodesically-critical value bigger than ϵ. If we were to increase ϵ' "infinitesimally," then, by definition, $\pi_u(\epsilon')$ would have at least one fewer edge. For this to occur, two consecutive edges of $\pi_u(\epsilon')$ would have to be replaced by a single edge. Thus, the vertex incident upon the two removed edges is a pinch vertex; hence ϵ' is in E. Now, let ϵ'' be the smallest value in E. Since there is no value in E smaller than ϵ'' in E, $\pi_u(\hat{\epsilon})$ does not change for $\hat{\epsilon} \in (\epsilon, \epsilon'']$. Thus, there is a global change to $\pi_u(\hat{\epsilon})$ for $\hat{\epsilon} > \epsilon''$, which implies that ϵ'' is geodesically-critical. Therefore, $\epsilon' = \epsilon''$, which completes the proof. □

Our method for maintaining $\pi_u(\epsilon)$, then, is as follows. We store the values belonging to E in a priority queue that supports the operations of insert, delete, and extract-min in $O(\log n)$ time (e.g., see [17]). While E is not empty, we extract the smallest $\hat{\epsilon}$ in E, perform the modification of $\pi_u(\hat{\epsilon})$ implied by this geodesically-critical value, and then update E to reflect the new geodesic path. This update involves examining the two vertices of $\pi_u(\hat{\epsilon})$ that now become adjacent and updating E accordingly. If either of them were previously pinch vertices, then we remove its corresponding locally-critical value from E. Likewise, if either of them becomes a pinch vertex after performing the update for $\hat{\epsilon}$, then we insert its new locally-critical value into E. Since we reduce by one the number of edges of the geodesic path with each event in E, the total number of events must be $O(n)$; hence, the total time to enumerate all the geodesically-critical values is $O(n \log n)$.

Given these geodesically-critical values it is then a simple manner to determine the consecutive pair that contains ϵ^* by using the method of Hershberger and Snoeyink [32], Guibas *et al.* [29], or Hakimi and Schmeichel [30] to drive a binary search among the set of geodesically-critical values. Using one of these simple algorithms, all of which are based upon the "greedy method," to test if a given ϵ is smaller than ϵ^* takes $O(n)$ time. This implies that we can determine, in $O(n \log n)$ time, the combinatorial structure of $\pi_u(\epsilon^*)$, and, as mentioned above, a similar procedure gives us the combinatorial structure of $\pi_g(\epsilon^*)$. Therefore, we can identify all the inflection edges in $H(\epsilon^*)$, each of which F must contain, by Lemma 2.1.

All that is left, then, is for us to determine all the links of F that do not contain inflection edges.

4 Completing the Construction

Whereas we used geodesic paths to find the zig-zag and zag-zig links, to complete the construction we use a related structure—the visibility graph. In particular, recall that the visibility graph of a set of line segments R has a vertex for each endpoint of a segment in R and an edge for each pair (p, q) such that the line segment $\overline{pq}$ does cross any segment in R, although we allow $\overline{pq}$ to intersect segment endpoints and even contain the segment $\overline{pq}$ if it is in R. It is well-known, for example, that geodesic paths always follow visibility graph edges[3]. In our case we are interested in the visibility graph defined on the segments in $U(\epsilon) \cup G(\epsilon)$. We say that an line segment s is *U-anchored* (resp., *G-anchored*) if s contains an edge e in the visibility graph of $U(\epsilon^*) \cup G(\epsilon^*)$ such that both of e's vertices lie on $U(\epsilon^*)$ (resp., $G(\epsilon^*)$). The following lemma establishes an important relationship between a best k-link function approximation to S and these anchored links.

Lemma 4.1: *Any canonical best k-link approximation to S has a U-anchored link containing a vertex of $G(\epsilon^*)$ or a G-anchored link containing a vertex of $U(\epsilon^*)$.*

Proof: The proof follows immediately from the characterization lemma of Hakimi and Schmeichel [30]. □

Our method for enumerating all such edges is based upon several applications of the parametric searching technique, as optimized by Cole [13, 14]. One such optimization applies to any situation that involves a set $Z = \{z_1, z_2, \ldots, z_m\}$ of m independent binary searches among an ordered set $A = (a_1, a_2, \ldots, a_n)$ of n items, where each comparison $c(a_i, z_j)$ is parameterized by ϵ. The outcome of $c(a_i, z_j)$ depends upon which of a constant number of intervals, determined by a_i and z_j, contain ϵ^*. If it takes T steps to determine if $\epsilon^* < \epsilon$, for a particular ϵ, then this parametric searching technique allows us to perform all m binary searches in $O((T + m)(\log n + \log m))$ time (see Cole [13] for details). It also gives us an interval $[\epsilon_1, \epsilon_2]$ containing ϵ^*, which is the intersection of all the intervals determined to contain ϵ^* during the m searches.

The second optimization applies when one wishes to sort a set $A = \{a_1, a_2, \ldots, a_n\}$, where, as in the previous case, each comparison $c(a_i, a_j)$ is parameterized by ϵ so that the outcome of $c(a_i, a_j)$ depends upon which of a constant number of intervals, determined by a_i and a_j, contain ϵ^*. If it takes T steps to determine if $\epsilon^* < \epsilon$, for a particular ϵ, then this parametric searching technique allows us to sort the elements of A in $O((T + n) \log n)$ time (see Cole [13, 14] for details). Also, this method gives an interval $[\epsilon_1, \epsilon_2]$ containing ϵ^*, which is the intersection of all the intervals determined to contain ϵ^* during the sort.

The challenge, of course, in applying these techniques is to design the parameterized sorts and searches so that the result is meaningful. Let us therefore turn to our application of these techniques.

4.1 Finding ϵ^*

Our method for completing the construction is to perform a parametric search for a U-anchored or G-anchored link satisfying Lemma 4.1. Observe that finding such a link effectively "clamps" $U(\epsilon)$ and $G(\epsilon)$ at $\epsilon = \epsilon^*$. Also note that we can use the linear-time method of Hershberger and Snoeyink [32], Guibas *et al.* [29], or Hakimi and Schmeichel [30] to resolve comparisons and to give us the final approximation F once we have narrowed the interval of candidate ϵ values to $[\epsilon^*, \epsilon^*]$.

To simplify the discussion let us concentrate on the problem of determining a U-anchored link that contains a vertex of $G(\epsilon^*)$; the method is similar for visibility edges anchored on $G(\epsilon)$. Our algorithm will actually "dovetail" the search for a U-anchored link containing a vertex of $G(\epsilon^*)$ with the search for a G-anchored link containing a vertex of $U(\epsilon^*)$.

Call a vertex p of $U(\epsilon)$ a *left inflection* (resp., *right inflection*) vertex if p is the left (resp., right) endpoint of an inflection edge of $H(\epsilon^*)$. Following an approach similar to that used by Ghosh [23], consider a portion of $U(\epsilon)$ between a left inflection vertex p and the leftmost

[3] For more information about visibility graphs and their properties see the excellent book by O'Rourke [50].

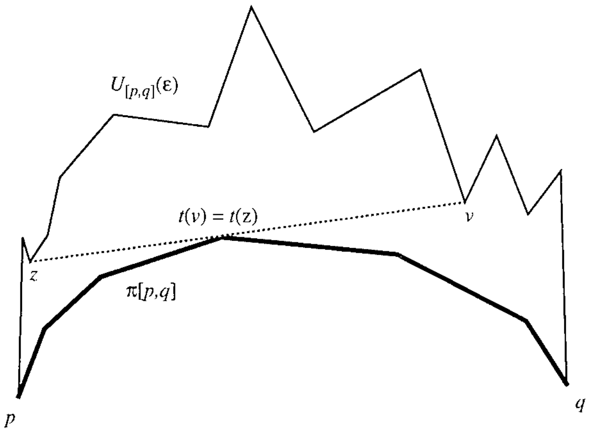


Figure 5: A U-anchored edge determining ϵ^*.

right inflection vertex q to the right of p. Denote this portion of $U(\epsilon)$ as $U_{[p,q]}(\epsilon)$. Note that the portion of $\pi_u(\epsilon^*)$ between p and q is a convex chain of edges such that each consecutive pair forms a "right turn." Denote this portion of $\pi_u(\epsilon^*)$ as $\pi[p,q]$ (but note that we have yet to determine ϵ^*—at this point we only know the combinatorial structure of $\pi_u(\epsilon^*)$). Finally, observe that if $U_{[p,q]}(\epsilon)$ contains the endpoints v and z of the U-anchored link that we seek, then $\overline{vz}$ must be equal to the the common tangent of v and $\pi[p,q]$ as well as z and $\pi[p,q]$. (See Figure 5.) Moreover, if the link satisfying Lemma 4.1 is a U-anchored link, then it must contain such a tangent edge for some $U_{[p,q]}(\epsilon)$.

Our first parametric search therefore is to determine for each vertex v on $U_{[p,q]}(\epsilon)$ its vertex of tangency, $t(v)$, with the convex chain $\pi[p,q]$ at $\epsilon = \epsilon^*$ (see Figure 5). In this case we can use binary-search-based parametric searching [13] applied to a well-known "binary search" tangent-finding method (e.g., see [21, 51]) to find all such tangents in $O(n \log n)$ time. This may still not restrict our interval of ϵ values to $[\epsilon^*, \epsilon^*]$, however, for a vertex v will have the same vertex of tangency, $t(v)$, over a range of ϵ values.

For each vertex w on $\pi[p,q]$, collect each vertex v on $U_{[p,q]}(\epsilon)$ such that $w = t(v)$ into two sets—$l(w)$, containing the vertices to the left of w, and $r(w)$, containing the vertices to the right of w. If w is the vertex of tangency for the U-anchored edge we seek, then that edge is determined by a $v \in l(w)$ and a $z \in r(w)$. In order for these two vertices to be able "search for each other," however, we must first order the vertices in $l(w)$ and $r(w)$ radially around w. To accomplish this we make a second application of parametric searching, this time using the second version, based upon a parallel sorting algorithm [13, 14], together with the linear-time method of Hershberger and Snoeyink [32], Guibas *et al.* [29], or Hakimi and Schmeichel [30] for resolving comparisons, to sort all $l(w)$ and $r(w)$ lists in $O(n \log n)$ time. This may still not restrict the range of ϵ values to $[\epsilon^*, \epsilon^*]$, however, for the vertices in an $l(w)$ and $r(w)$ may have the same ordering with respect to w over a range of ϵ values.

Therefore, to complete the process we must perform one more application of parametric searching where we perform a binary search in $l(w)$ for each $z \in r(w)$ to locate the vertex in $l(w)$ hit by the ray $\overrightarrow{zt(z)}$, if it exists. Given that each $l(w)$ is now sorted, we can perform all of these binary-search parametric searches in $O(n \log n)$ time. Since one of these searches (or a corresponding search for G-anchored edges) must succeed, we finally are guaranteed to have restricted the range of candidate ϵ values to the interval $[\epsilon^*, \epsilon^*]$. Moreover, seeing how the last comparison in this parametric search was resolved using a minimum-link ordered stabber algorithm, this completes the construction, giving us the following theorem.

Theorem 4.2: *Given a set S of n points in the plane, and an integer parameter $1 \leq k \leq n-1$, one can construct a best k-link approximation to S under the uniform metric in $O(n \log n)$ time.*

5 Simplifying the Implementation

Although our method for finding a best k-link approximation to S is conceptually simple at a high level, implementing it in practice would probably not be so easy. The chief difficulty arises from the (up to) three calls to parametric searching that are made at the end of the algorithm; all the other steps would be relatively straightforward to implement. But those calls to parametric searching, especially the version based upon a parametric sorting, would be a challenge to implement, and the constant factors they would imply in the running time would not be very small. Fortunately, as we show in this section, we can considerably simplify these calls to parametric searching. In particular, we show how to eliminate the (second) call based upon sorting all together, and we observe how one can simplify all our calls to parametric searching through a simple randomization technique. The resulting algorithm will still be guaranteed to find a best k-link approximation to S, however, for the randomization will only impact the running time of our method, not its correctness. Nevertheless, it will result in a method that runs in $O(n \log n)$ time with very high probability. Moreover, the (expected) constant factors will be reasonably small.

Let us begin by describing how we can eliminate the call to sorting-based parametric searching, which

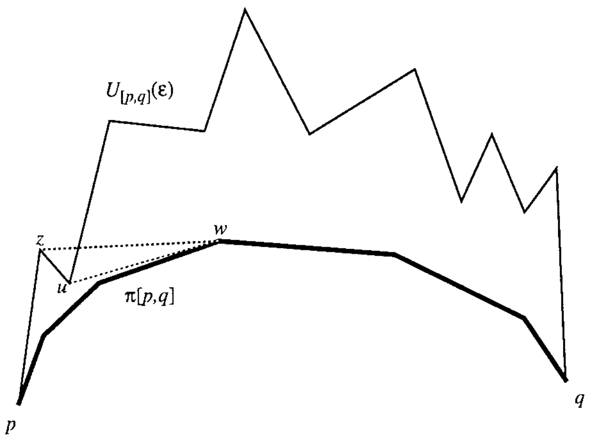


Figure 6: An illustration of why $l_{\text{vis}}(w)$ is already sorted.

comprises the second call to parametric searching in our algorithm. Recall that after we have made our first call to (binary-search based) parametric searching, we have determined for every vertex v on $U_{[p,q]}(\epsilon)$ its tangent, $t(v)$ on $\pi[p,q]$. Previously, we then collected, for each w on $\pi[p,q]$, all the vertices on $U_{[p,q]}(\epsilon)$ whose tangent is w to the "left set" $l(w)$ and the "right set" $r(w)$, and we sorted the vertices of $l(w)$ radially around w (using sorting-based parametric searching) so that we can then do a binary search in $l(w)$ for each vertex in $r(w)$ (using binary-search based parametric searching). Observe, however, that we do not need all the vertices of $l(w)$ to perform this search; we need only those vertices that are visible from w on $U_{[p,q]}(\epsilon^*)$. Thus, let us use $l_{\text{vis}}(w)$ to denote the set of vertices in $l(w)$ that are visible from w on $U_{[p,q]}(\epsilon^*)$. If we can determine these visible vertices, then, as the following lemma shows, we can sort them without resorting to parametric searching.

Lemma 5.1: *Sorting the vertices of $l_{\text{vis}}(w)$ radially around w as they appear on $U_{[p,q]}(\epsilon^*)$ gives the same order as a listing of the vertices of $l_{\text{vis}}(w)$ by increasing x-coordinates.*

Proof: Suppose, for the sake of a contradiction, that there are two vertices u and z in $l_{\text{vis}}(w)$ such that u is before z in a radial listing around w as they appear on $U_{[p,q]}(\epsilon^*)$, but $x(z) < x(u)$. (See Figure 6.) Since w is below the curve $U_{[p,q]}(\epsilon^*)$, this implies that $U_{[p,q]}(\epsilon^*)$ intersects the segment $\overline{zw}$. But this contradicts the fact that z is visible from w; hence, this establishes the lemma. □

Thus, it is sufficient for us to determine the members of $l_{\text{vis}}(w)$, for each w on $\pi[p,q]$, since they are already given by increasing x-coordinates (for that is how the vertices are ordered in S).

So, let us turn to the problem of determining the members of $l_{\text{vis}}(w)$ for some given w on $\pi[p,q]$. Let w' denote the vertex on $U_{[p,q]}(\epsilon)$ directly above w (i.e., if $w = g_i$, then $w' = u_i$). A simple consequence of the proof of Lemma 5.1 is that a vertex v in $l(w)$ is in $l_{\text{vis}}(w)$ if and only if the geodesic path from v to w' is above the line segment $\overline{vw}$. Indeed, it is sufficient that the first edge $e(v)$ in this path be above the segment $\overline{vw}$. Since each such geodesic path remains unchanged for $\epsilon \in [\epsilon_1, \epsilon_2]$ (because it is a sequence of left turns that occur at vertices of $U_{[p,q]}(\epsilon)$), we can determine $e(v)$ for each such v without knowing the value of ϵ^*. Specifically, we can use the data structure of Guibas and Hershberger [27], as simplified by Hershberger [31], to determine each $e(v)$ in $O(\log n)$ time. This data structure is relatively simple to construct and query, especially since each $U_{[p,q]}(\epsilon)$ is a single monotone chain.

Note that each $e(v)$ determines a critical $\hat{\epsilon}_v$ value such that v is visible if $\epsilon \geq \hat{\epsilon}$, and v is not visible otherwise. Thus, once we have determined all the $e(v)$'s, we can then determine which v's are visible from their respective $w = t(v)$ vertices simply by resolving all of the $\hat{\epsilon}$'s against ϵ^* using the method of Hershberger and Snoeyink [32], Guibas *et al.* [29], or Hakimi and Schmeichel [30] to drive a binary search. Since there are $O(n)$ such parameterized comparisons, and each resolution requires $O(n)$ time, the total time needed to resolve all of these comparisons is $O(n \log n)$. This implies that we can substitute the most simple form of parametric searching for the more-complicated sorting-based method [13, 14]. The resulting algorithm would still run in $O(n \log n)$ time. The only slightly impractical steps in this algorithm are repeated computations of medians (e.g., of the remaining unresolved $\hat{\epsilon}$'s in each iteration of the above binary search), for which the best deterministic algorithm has a relatively large constant (e.g., see [17]).

But the computation of a median or weighted median (which is the least-efficient step in the binary-search based parametric searching method of Cole [13]) can be significantly improved in practice through the use of randomization (e.g., see [17]). The effect in our case is that the resulting algorithm will still be guaranteed to find a best k-link approximation, but it will not be guaranteed to run in $O(n \log n)$ time. Nevertheless, it will run in $O(n \log n)$ time with very high probability, and the (expected) constants factors in the running time will be very reasonable. Thus, we suggest such a use of randomization in any implementation of our method.

6 Conclusion and Directions for Future Work

We have given an $O(n \log n)$-time method for finding a best k-link approximation to set of n points in the plane under the uniform metric, and we have even given a method for efficiently implementing it in practice. This suggests a number of interesting directions for future work. Some possibilities include examining metrics other than the uniform metric, approximating with higher-degree spline functions, and approximating more-general types of geometric objects (such as segments joined in a polygonal path). As with our results, the general goal should be to find a best k-piece approximation, since this is the version of the problem driven by bounds placed upon the computational resources needed to represent the approximation.

Acknowledgements

I would like to thank Jack Snoeyink for some stimulating discussions regarding minimum-link function approximations, for several helpful comments regarding earlier versions of this paper, and, in particular, for pointing out the counter-example illustrated in Figure 2. Finally, I would like to thank Esther Arkin, Mikhail Atallah, Paul Chew, Simon Kasif, S. Rao Kosaraju, Joe Mitchell, and David Mount for several helpful discussions concerning this and related problems.

References

[1] P. K. Agarwal, A. Aggarwal, B. Aronov, S. R. Kosaraju, B. Schieber, and S. Suri, "Computing external farthest neighbors for a simple polygon," *Discrete Appl. Math.*, **31**(2), 97–111, 1991.

[2] P. K. Agarwal, B. Aronov, M. Sharir, and S. Suri, "Selecting distances in the plane," in *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, 321–331, 1990.

[3] P. K. Agarwal and J. Matoušek, "Ray shooting and parametric search," in *Proc. 24th Annu. ACM Sympos. Theory Comput.*, 517–526, 1992.

[4] P. K. Agarwal, M. Sharir, and S. Toledo, "Applications of parametric searching in geometric optimization," in *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, 72–82, 1992.

[5] A. Aggarwal, H. Booth, J. O'Rourke, S. Suri, and C. K. Yap, "Finding minimal convex nested polygons," in *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, 296–304, 1985.

[6] E. M. Arkin, J. S. B. Mitchell, and S. Suri, "Optimal link path queries in a simple polygon," in *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, 269–279, 1992.

[7] B. Aronov, "On the geodesic Voronoi diagram of point sites in a simple polygon," *Algorithmica*, **4**, 109–140, 1989.

[8] R. E. Bellman and R. S. Roth, "Curve fitting by segmented straight lines," *American Statistical Association Journal*, **64**, 1079–1084, 1969.

[9] R. E. Bellman and R. S. Roth, *Methods in Approximation: Techniques for Mathematical Modelling*, D. Reidel Publishing Co., Boston, 1986.

[10] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink, "Ray shooting in polygons using geodesic triangulations," in *Proc. 18th Internat. Colloq. Automata Lang. Program., Lecture Notes in Computer Science*, vol. 510, Springer-Verlag, 661–673, 1991.

[11] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, "Diameter, width, closest line pair, and parametric searching," in *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, 120–129, 1992.

[12] K. L. Clarkson, "Linear programming in $O(n3^{d^2})$ time," *Inform. Process. Lett.*, **22**, 21–24, 1986.

[13] R. Cole, "Slowing down sorting networks to obtain faster sorting algorithms," *J. ACM*, **34**, 200–208, 1987.

[14] R. Cole, "Parallel merge sort," *SIAM J. Comput.*, **17**(4), 770–785, 1988.

[15] R. Cole, J. Salowe, W. Steiger, and E. Szemerédi, "An optimal-time algorithm for slope selection," *SIAM J. Comput.*, **18**, 792–810, 1989.

[16] S. D. Conte and C. de Boor, *Elementary Numerical Analysis: An Algorithmic Approach*, McGraw-Hill, New York, 3rd edition, 1980.

[17] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Mass., 1990.

[18] P. J. Davis, *Interpolation and Approximation*, Blaisdell Publishing Co., New York, 1963.

[19] P. Dierckx, *Curve and Surface Fitting with Splines*, Clarendon Press, New York, 1993.

[20] M. E. Dyer, "Linear time algorithms for two- and three-variable linear programs," *SIAM J. Comput.*, **13**, 31–45, 1984.

[21] H. Edelsbrunner, *Algorithms in Combinatorial Geometry, EATCS Monographs on Theoretical Computer Science*, vol. 10, Springer-Verlag, Heidelberg, West Germany, 1987.

[22] H. ElGindy and M. T. Goodrich, "Parallel algorithms for shortest path problems in polygons," *Visual Comput.*, **3**, 371–378, 1988.

[23] S. Ghosh, "Computing visibility polygon from a convex set and related problems," *J. Algorithms*, **12**, 75–95, 1991.

[24] S. K. Ghosh and A. Maheshwari, "Parallel algorithms for all minimum link paths and link center problems," in *Proc. 3rd Scand. Workshop Algorithm Theory, Lecture Notes in Computer Science*, vol. 621, Springer-Verlag, 106–117, 1992.

[25] M. T. Goodrich and R. Tamassia, "Dynamic ray shooting and shortest paths via balanced geodesic triangulations," in *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 318–327, 1993.

[26] L. J. Guibas and J. Hershberger, "Optimal shortest path queries in a simple polygon," in *Proc. 3rd Annu. ACM Sympos. Comput. Geom.*, 50–63, 1987.

[27] L. J. Guibas and J. Hershberger, "Optimal shortest path queries in a simple polygon," *J. Comput. Syst. Sci.*, **39**, 126–152, 1989.

[28] L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan, "Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons," *Algorithmica*, **2**, 209–233, 1987.

[29] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink, "Approximating polygons and subdivisions with minimum link paths," in *Proc. 2nd Annu. SIGAL Internat. Sympos. Algorithms, Lecture Notes in Computer Science*, vol. 557, Springer-Verlag, 151–162, 1991.

[30] S. L. Hakimi and E. F. Schmeichel, "Fitting polygonal functions to a set of points in the plane," *CVGIP: Graph. Mod. Image Proc.*, **53**(2), 132–136, 1991.

[31] J. Hershberger, "A new data structure for shortest path queries in a simple polygon," *Inform. Process. Lett.*, **38**, 231–235, 1991.

[32] J. Hershberger and J. Snoeyink, "Computing minimum length paths of a given homotopy class," in *Proc. 2nd Workshop Algorithms Data Struct., Lecture Notes in Computer Science*, vol. 519, Springer-Verlag, 331–342, 1991.

[33] H. Imai and M. Iri, "Computational-geometric methods for polygonal approximations of a curve," *Comput. Vision Graph. Image Process.*, **36**, 31–41, 1986.

[34] H. Imai and M. Iri, "An optimal algorithm for approximating a piecewise linear function," *Journal of Information Processing*, **9**(3), 159–162, 1986.

[35] H. Imai and M. Iri, "Polygonal approximations of a curve-formulations and algorithms," in *Computational Morphology*, G. T. Toussaint, editor, North-Holland, Amsterdam, Netherlands, 71–86, 1988.

[36] D. L. B. Jupp, "Approximation to data by splines with free knots," *SIAM Journal on Numerical Analysis*, **15**(2), 328–343, 1978.

[37] Y. Ke, "An efficient algorithm for link-distance problems," in *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, 69–78, 1989.

[38] D. T. Lee and F. P. Preparata, "Euclidean shortest paths in the presence of rectilinear barriers," *Networks*, **14**, 393–410, 1984.

[39] W. Lenhart, R. Pollack, J.-R. Sack, R. Seidel, M. Sharir, S. Suri, G. T. Toussaint, S. Whitesides, and C. K. Yap, "Computing the link center of a simple polygon," *Discrete Comput. Geom.*, **3**, 281–293, 1988.

[40] J. Matoušek, M. Sharir, and E. Welzl, "A subexponential bound for linear programming," in *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, 1–8, 1992.

[41] N. Megiddo, "Applying parallel computation algorithms in the design of serial algorithms," *J. ACM*, **30**, 852–865, 1983.

[42] N. Megiddo, "Linear-time algorithms for linear programming in R^3 and related problems," *SIAM J. Comput.*, **12**, 759–776, 1983.

[43] N. Megiddo, "Linear programming in linear time when the dimension is fixed," *J. ACM*, **31**, 114–127, 1984.

[44] A. Melkman and J. O'Rourke, "On polygonal chain approximation," in *Computational Morphology*, G. T. Toussaint, editor, North-Holland, Amsterdam, Netherlands, 87–95, 1988.

[45] J. S. B. Mitchell, "An algorithmic approach to some problems in terrain navigation," *Artif. Intell.*, **37**, 171–201, 1988.

[46] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The discrete geodesic problem," *SIAM J. Comput.*, **16**, 647–668, 1987.

[47] J. S. B. Mitchell, C. Piatko, and E. M. Arkin, "Computing a shortest k-link path in a polygon," in *Proc. 33rd Annu. IEEE Sympos. Found. Comput. Sci.*, 573–582, 1992.

[48] J. S. B. Mitchell, G. Rote, and G. Woeginger, "Minimum-link paths among obstacles in the plane," in *Proc. 6th Annu. ACM Sympos. Comput. Geom.*, 63–72, 1990.

[49] J. S. B. Mitchell and S. Suri, "Separation and approximation of polyhedral surfaces," in *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, 296–306, 1992.

[50] J. O'Rourke, *Art Gallery Theorems and Algorithms*, Oxford University Press, New York, NY, 1987.

[51] F. P. Preparata and M. I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, New York, NY, 1985.

[52] J.-M. Robert and G. Toussaint, "Linear approximation of simple objects," in *Proc. 9th Sympos. Theoret. Aspects Comput. Sci., Lecture Notes in Computer Science*, vol. 577, Springer-Verlag, 233–244, 1992.

[53] R. Seidel, "Small-dimensional linear programming and convex hulls made easy," *Discrete Comput. Geom.*, **6**, 423–434, 1991.

[54] S. Suri, "A linear time algorithm for minimum link paths inside a simple polygon," *Comput. Vision Graph. Image Process.*, **35**, 99–110, 1986.

[55] S. Suri, *Minimum link paths in polygons and related problems*, Ph.D. thesis, Dept. Comput. Sci., Johns Hopkins Univ., Baltimore, MD, 1987.

[56] S. Suri, "Computing geodesic furthest neighbors in simple polygons," *J. Comput. Syst. Sci.*, **39**, 220–235, 1989.

[57] G. T. Toussaint, "On the complexity of approximating polygonal curves in the plane," in *Proc. IASTED, International Symposium on Robotics and Automation*, Lugano, Switzerland, 1985.

[58] D. P. Wang, N. F. Huang, H. S. Chao, and R. C. T. Lee, "Plane sweep algorithms for polygonal approximation problems with applications," in *Proc. 4th Annu. Internat. Sympos. Algorithms Comput. (ISAAC 93), Lecture Notes in Computer Science*, vol. 762, Springer-Verlag, 515–522, 1993.

Applications of Weighted Voronoi Diagrams and Randomization to Variance-Based k-Clustering

(Extended Abstract)

Mary Inaba*, Naoki Katoh† and Hiroshi Imai*

Abstract

In this paper we consider the k-clustering problem for a set S of n points $p_i = (\boldsymbol{x}_i)$ in the d-dimensional space with variance-based errors as clustering criteria, motivated from the color quantization problem of computing a color lookup table for frame buffer display. As the inter-cluster criterion to minimize, the sum of intra-cluster errors over every cluster is used, and as the intra-cluster criterion of a cluster S_j,

$$|S_j|^{\alpha-1} \sum_{p_i \in S_j} \|\boldsymbol{x}_i - \bar{\boldsymbol{x}}(S_j)\|^2$$

is considered, where $\|\cdot\|$ is the L_2 norm and $\bar{\boldsymbol{x}}(S_j)$ is the centroid of points in S_j, i.e., $(1/|S_j|)\sum_{p_i \in S_j} \boldsymbol{x}_i$. The cases of $\alpha = 1, 2$ correspond to the sum of squared errors and the all-pairs sum of squared errors, respectively.

The k-clustering problem under the criterion with $\alpha = 1, 2$ are treated in a unified manner by characterizing the optimum solution to the k-clustering problem by the ordinary Euclidean Voronoi diagram and the weighted Voronoi diagram with both multiplicative and additive weights. With this framework, the problem is related to the generalized primary shutter function for the Voronoi diagrams. The primary shutter function is shown to be $O(n^{O(kd)})$, which implies that, for fixed k, this clustering problem can be solved in a polynomial time. For the problem with the most typical intra-cluster criterion of the sum of squared errors, we also present an efficient randomized algorithm which, roughly speaking, finds an ϵ-approximate 2-clustering in $O(n(1/\epsilon)^d)$ time, which is quite practical and may be used to real large-scale problems such as the color quantization problem.

1 Introduction

Clustering is the grouping of similar objects and a clustering of a set is a partition of its elements that is chosen to minimize some measure of dissimilarity. It is very fundamental and used in various fields in computer science such as pattern recognition, learning theory, image processing and computer graphics. There are various kinds of measure of dissimilarity, called criteria, in compliance with the problem.

Definition of the k-clustering problem: The general k-clustering problem can be defined as follows. A k-clustering is a partition of the given set S of n points $p_i = (\boldsymbol{x}_i)$ $(i = 1, \ldots, n)$ in the d-dimensional space into k disjoint nonempty subsets $S_1, \ldots, S_k$, called clusters. A k-clustering is measured by the following two criteria.

(Intra-cluster criterion) For each cluster S_j, the measure (or error) Intra(S_j) of S_j, representing how good the cluster S_j is, is defined appropriately by applications. Typical intra-cluster criteria are the diameter, radius, variance, variance multiplied by $|S_j|$ (sum of squared errors) and variance multiplied by $|S_j|^2$ (all-pairs sum of squared errors) of point set S_j.

(Inter-cluster criterion) The inter-cluster criterion defines the total cost of the k-clustering, which is a function of Intra(S_j) $(j = 1, \ldots, k)$ and is denoted by Inter($y_1, y_2, \ldots, y_k$) where y_j = Intra(S_j). Typical function forms are $\max\{\, y_j \mid j = 1, \ldots, k\,\}$ and $\sum_{i=1}^{k} y_k$.

Then, the k-clustering problem is to find a k-clustering which minimizes the inter-cluster criterion:

$$\min\{\, \text{Inter}(\text{Intra}(S_1), \ldots, \text{Intra}(S_k)) \mid k\text{-clustering } (S_1, \ldots, S_k) \text{ of } S\,\}$$

Previous results concerning diameter and radius: In computational geometry, many results have

[*] Department of Information Science, University of Tokyo, Tokyo 113, Japan

[†] Department of Management Science, Kobe University of Commerce, Kobe 651-21, Japan

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

been obtained for the clustering problem. The diameter and radius problems are rather well studied. They include an $O(n \log n)$-time algorithm for finding a 2-clustering of n points in the plane which minimizes the maximum diameter (Asano, Bhattacharya, Keil and Yao [1]), an $O(n^2 \log^2 n)$-time algorithm for finding a 3-clustering of planar point set which minimizes the maximum diameter (Hagauer and Rote [5]), and an $O(n \log^2 n / \log\log n)$-time algorithm for finding a 2-clustering which minimizes the sum of the two diameters (Hershberger [10]). When k is regarded as a variable, most k-clustering problems become NP-hard (e.g., see Megiddo and Supowit [14], Feder and Greene [4]). For fixed k, the k-clustering problem using the diameter and radius as the intra-cluster criterion and a monotone function, including taking the maximum and the summation, as the inter-cluster criterion can be solved in a polynomial time (Capoyleas, Rote and Woeginger [3]).

There are also proposed approximate algorithms for the diameter and radius whose approximation ratio is theoretically guaranteed. Feder and Greene [4] gave optimal approximate algorithms whose running time is $O(n \log k)$ for n points in the d-dimensional space for fixed d, and whose worst-case ratio is 2.

Motivation for the variance-based clustering: In this paper, we consider the k-clustering problem with variance-based measures as an intra-cluster criterion. This is motivated from the color quantization problem of computing a color lookup table for frame buffer display. Typical color quantization problems cluster hundreds of thousands of points in the RGB three-dimensional space into $k = 256$ clusters. Since k is large, a top-down approach to recursively divide the point set into 2 clusters is mostly employed. In this problem, the diameter and radius are not suited as an intra-cluster criterion, and the variance-based (Wan, Wong and Prusinkiewicz [15]) and L_1-based (median cut; Heckbert [9]) criteria are often used. In [15], [9], the top-down approach is used and in solving the 2-clustering problem both only treat separating planes orthogonal to some coordinate axis. These algorithms are implemented in `rlequant` of Utah Raster Toolkit, and `ppmquant` of X11R5 or `tiffmedian` of Tiff Soft. Although these implementations run rather fast in practice, roughly speaking in $O(n \log n)$ time, there is no theoretical guarantee about how good their solution k-clusterings are.

Rigorous definition of the variance-based clustering: Therefore, it is required to develop a fast 2-clustering algorithm and to determine the complexity of the k-clustering problem for the variance-based case. Before describing the existing computational-geometric results concerning variance-based case, let us define the variance-based intra-cluster criterion in a rigorous way. The variance $\mathrm{Var}(S)$ of points $p_i = (\boldsymbol{x}_i)$ in S is defined by

$$\mathrm{Var}(S) = \frac{1}{|S|} \sum_{p_i \in S} \|\boldsymbol{x}_i - \bar{\boldsymbol{x}}(S)\|^2$$

where $\bar{\boldsymbol{x}}(S)$ is the centroid of S:

$$\bar{\boldsymbol{x}}(S) = \frac{1}{|S|} \sum_{p_i \in S} \boldsymbol{x}_i.$$

For a parameter α, define $\mathrm{Var}^\alpha(S)$ by

$$\mathrm{Var}^\alpha(S) = |S|^\alpha \mathrm{Var}(S).$$

Var^0 is exactly the variance itself. Var^1 is represented as

$$\mathrm{Var}^1(S) = \sum_{p_i \in S} \|\boldsymbol{x}_i - \bar{\boldsymbol{x}}(S)\|^2$$

and hence is the sum of squared errors with respect to the centroid of S. Var^2 is represented as

$$\begin{aligned}\mathrm{Var}^2(S) &= |S| \sum_{p_i \in S} \|\boldsymbol{x}_i - \bar{\boldsymbol{x}}(S)\|^2 \\ &= \sum_{p_i, p_l \in S,\ i<l} \|\boldsymbol{x}_i - \boldsymbol{x}_l\|^2\end{aligned}$$

and hence is the all-pairs sum of squared errors in S. Adopting Var^α as the intra-cluster metric, as α becomes larger, the sizes of clusters in an optimum k-clustering becomes more balanced.

Previous results on the variance-based clustering: For the variance-based criteria, unlike the diameter and radius, the k-clustering problem adopting the maximum function as the inter-cluster criterion becomes hard to solve. For this inter-cluster criterion with the all-pairs sum of squared errors, only a pseudo-polynomial approximation scheme is known (Hasegawa, Imai, Inaba, Katoh and Nakano [7]). Also, in applications such as the color quantization problem, the summation function is adopted as an inter-cluster criterion [15]. In this paper, we consider only the summation case, that is, the k-clustering problem to minimize the summation of variance-based intra-cluster costs over clusters.

For the variance-based clustering problem with the summation function as an inter-cluster metric, the following are known. Concerning Var^1, the sum of squared errors, it is well known that an optimum 2-clustering is linearly separable and that an optimum k-clustering is induced by the Voronoi diagram generated by k points (e.g., see [2, 7, 15]). Using this characterization together with standard computational-geometric techniques, the 2-clustering problem with Var^1 as the intra-cluster metric can be solved in $O(n^2)$ time and $O(n)$ space, and the k-clustering problem is solvable in a polynomial time when k is fixed [7]. Concerning Var^2, the all-pairs sum of squared errors, an optimum 2-clustering is circularly separable (Boros and Hammer [2]), and a finer characterization by using the higher-order Voronoi diagram is given in [7]. Using this characterization, the 2-clustering problem with Var^2 as the intra-cluster metric can be solved in $O(n^{d+1})$ time, and also it is seen that the k-clustering problem for this case can be solved in a polynomial time $O(n^{(d+1)k(k-1)/2})$ when k is fixed [6].

There is also proposed an approximate algorithm for the k-clustering problem with Var^1 as an intra-cluster metric. Hasegawa, Imai, Inaba, Katoh and Nakano [7] gave an $O(n^{k+1})$-time algorithm for fixed d whose worst-case ratio is 2. This algorithm solve the k-clustering problem with constraining the representative point of each cluster to be one of points in the cluster.

Our results: In this paper, the k-clustering problem under the intra-cluster criterion Var^α with $\alpha = 1, 2$ is treated in a unified way by characterizing the optimum solution to the k-clustering problem by the ordinary Voronoi diagram and the weighted Voronoi diagrams with both multiplicative and additive weights.

With this framework, the problem is related to the generalized primary shutter function for the Voronoi diagrams, which is roughly the number of partitions of n points in the d-dimensional space induced by the Voronoi diagram generated by k generator points. The primary shutter function of the Euclidean Voronoi diagram is shown to be $O(n^{dk})$, and that for the Voronoi diagram with additive and multiplicative weights $O(n^{(d+2)k})$. Based on these, the k-clustering problem for n points in the d-dimensional space with a variance-based criterion can be solved in $O(n^{(d+2)k+1})$ time. This greatly improves the previous bound $O(n^{O(dk^2)})$. We have thus given a polynomial-time algorithm for the case of fixed k, but its degree is large even for moderate values of d and k.

To develop a practically useful 2-clustering algorithm, for the problem with the most typical intra-cluster criterion of the sum of squared errors, we present an efficient randomized algorithm which, roughly speaking, finds an ϵ-approximate 2-clustering in $O(n(1/\epsilon)^d)$ time, which is quite practical and may be used to real large-scale problems such as the color quantization problem. In the analysis, a fact that this intra-cluster cost has its statistical meanings by definition is used. This randomized algorithm can be easily generalized to the k-clustering problem.

2 A unified approach to the variance-based k-clustering by weighted Voronoi diagrams

The variance-based k-clustering problem is described as follows:

$$\min\{\sum_{j=1}^{k} \mathrm{Var}^\alpha(S_j) \mid k\text{-clustering } (S_1,\ldots,S_k) \text{ of } S\}$$

In [7], a parametric characterization was given for the case of $\alpha = 2$ (all-pairs case) by using a general parametric technique for minizing quasiconcave functions developed by Katoh and Ibaraki [12], which enabled us to characterize an optimal 2-clustering for $\alpha = 2$ by means of higher-order Voronoi diagram, and to obtain a pseudo polynomial approximation scheme for the 2-clustering problem for Var^2 and the maximum function as the inter-cluster metric.

In this paper, we concentrate on the case where the summation function is adopted for the inter-cluster criterion, and give a more direct characterization for the problem with $\alpha = 1, 2$.

We may make use of partitions of n points induced by weighted Voronoi diagrams. Consider k points $q_j = (\boldsymbol{\mu}_j)$ in the d-dimensional space with multiplicative weight ν_j and additive weight σ_j $(j = 1, \ldots, k)$. Define the Voronoi region $\mathrm{Vor}(q_j)$ of q_j by

$$\mathrm{Vor}(q_j) = \bigcap_{l=1}^{k}\{p = (\boldsymbol{x}) \mid \nu_j\|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2 + \sigma_j \le \nu_l\|\boldsymbol{x} - \boldsymbol{\mu}_l\|^2 + \sigma_l\}$$

For any point in $\mathrm{Vor}(q_j)$, q_j is the closest point among q_l $(l = 1, \ldots, k)$ with respect to the weighted distance. $\mathrm{Vor}(q_j)$ $(j = 1, \ldots, k)$ partitions the space, which is called the *weighted Voronoi diagram* generated by these k points q_j. When $\sigma_j = 0$ and $\nu_j = 1$ $(j = 1, \ldots, k)$, this weighted Voronoi diagram reduces to the ordinary Euclidean Voronoi diagram.

By the Voronoi diagram generated by these k weighted points, n points in the given set S are naturally partitioned into k clusters (we here ignore the case in this definition where a point in S is equidistant from two points among these k weighted points). We call this partition a *Voronoi partition* of n points in S by k weighted generators. Apparently, not all k-clusterings are Voronoi partitions. In fact, we can characterize optimal k-clusterings by the Voronoi partition. The case of $\alpha = 1$ is well known, and we here state only the theorem.

Theorem 1 ([2, 7, 15]) *Suppose that $(S_1^*,\ldots,S_k^*)$ is an optimum k-clustering for the k-clustering problem with* Var^1 $(\alpha = 1)$ *as the intra-cluster metric. Then, an optimum k-clustering is a Voronoi partition by the ordinary Euclidean Voronoi diagram for k points* $q_j = (\boldsymbol{\mu}_j^*)$ $(\boldsymbol{\mu}_j^* = \bar{\boldsymbol{x}}(S_j^*))$. □

Now, we prove the following theorem for the case of Var^2, i.e., all-pairs sum of squared errors.

Theorem 2 *Suppose that $(S_1^*,\ldots,S_k^*)$ is an optimum k-clustering for the k-clustering problem with* Var^2 $(\alpha = 2)$ *as the intra-cluster metric. Then, an optimum k-clustering is a Voronoi partition by the weighted Voronoi diagram for k points $q_j = (\boldsymbol{\mu}_j^*)$ $(\boldsymbol{\mu}_j^* = \bar{\boldsymbol{x}}(S_j^*))$ with multiplicative weight $\nu_j^* = |S_j^*|$ and additive weight σ_j defined by $\sigma_j = \sum_{p_i \in S_j^*} \|\boldsymbol{x}_i - \bar{\boldsymbol{x}}(S_j^*)\|^2$.*

Proof: First, observe the following relation.

$$\begin{aligned}
&\sum_{p_i \in S_j} \|\boldsymbol{x}_i - \boldsymbol{x}\|^2 \\
&\quad = \sum_{p_i \in S_j} \|(\boldsymbol{x}_i - \bar{\boldsymbol{x}}(S_j)) + (\bar{\boldsymbol{x}}(S_j) - \boldsymbol{x})\|^2 \\
&\quad = |S_j| \cdot \|\boldsymbol{x} - \bar{\boldsymbol{x}}(S_j)\|^2 + \sum_{p_i \in S_j} \|\boldsymbol{x}_i - \bar{\boldsymbol{x}}(S_j)\|^2
\end{aligned}$$

where it should be noted that $\sum_{p_i \in S_j}(\boldsymbol{x}_i - \bar{\boldsymbol{x}}(S_j)) = 0$.

Now, suppose that, in the weighted Voronoi diagram above, a point $p_i \in S_j^*$ is not contained in $\mathrm{Vor}(q_j)$ for $q_j = (\bar{\boldsymbol{x}}(S_j^*))$, and is in $\mathrm{Vor}(q_{j'})$. Then, moving p_i from S_j^* to $S_{j'}^*$, the total cost is strictly reduced from the above formula (note that Var^2 is the all-pairs sum of squared errors), which contradicts the optimality of $(S_1^*,\ldots,S_k^*)$. Hence, each $p_j \in S_j^*$ is contained in $\mathrm{Vor}(q_j)$ for $j = 1,\ldots,k$, and the theorem follows. □

By Theorem 1 and Theorem 2, the variance-based k-clustering problem with $\alpha = 1, 2$ can be solved by enumerating all the Voronoi partitions of n points generated by k weighted points, and finding a partition with minimum one.

The number of distinct Voronoi partitions has strong connection with the generalized primary shutter function for k-label space introduced by Hasegawa [6, 8] which has applications in computational learning theory. We here analyze this number by showing that Voronoi partitions are duals of arrangements of algebraic surfaces in the $O(dk)$-dimensional space.

Theorem 3 *The number of Voronoi partitions of n points by the Euclidean Voronoi diagram generated by k points in the d-dimensional space is $O(n^{dk})$, and all the Voronoi partitions can be enumerated in $O(n^{dk+1})$ time.*

Proof: In the ordinary Voronoi diagram, all multiplicative weights are one and all additive weights are zero, i.e., $\nu_j = 1$, $\sigma_j = 0$ $(j = 1,\ldots,k)$. Parameters are $\boldsymbol{\mu}_j$ $(j = 1,\ldots,k)$. Consider the (dk)-dimensional vector space consisting of $\boldsymbol{\mu}$ with $\boldsymbol{\mu} = (\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k)$. In this (dk)-dimensional space, we can define an equivalence relation among points such that two points are in the equivalence relation if their corresponding Voronoi partitions are identical. The equivalence relation produces a subdivision of this space into equivalence classes.

For each pair of distinct $\boldsymbol{\mu}_{j_1}$ and $\boldsymbol{\mu}_{j_2}$ among $\boldsymbol{\mu}_j$ $(j = 1,\ldots,k)$ and each point $p_i = (\boldsymbol{x}_i)$ among p_i $(i = 1,\ldots,n)$, consider an algebraic surface in this dk-dimensional space defined by

$$\|\boldsymbol{x}_i - \boldsymbol{\mu}_{j_1}\|^2 - \|\boldsymbol{x}_i - \boldsymbol{\mu}_{j_2}\|^2 = 0$$

where $\boldsymbol{x}_i$ is regarded as a constant vector. The number of such surfaces is $nk(k-1)/2$. The arrangement of these $nk(k-1)/2$ algebraic surfaces coincides with the subdivision defined by the equivalence relation from Voronoi partitions. The number of Voronoi partitions is bounded by the combinatorial complexity of the arrangement of $nk(k-1)/2$ constant-degree algebraic surfaces, and the theorem follows. □

A further detailed analysis about the primary shutter function of the k-Voronoi space is done by Ishiguro [11]. In that paper, the linearization technique is applied in the analysis, and an algorithm using hyperplane arrangements is given based on it. Also, an application of the primary shutter function to learning an unknown k-Voronoi space from examples is shown.

Theorem 4 *The number of Voronoi partitions of n points by the weighted Voronoi diagram generated by k weighted points with a multiplicative weight and an additive weight in the d-dimensional space is $O(n^{(d+2)k})$, and all the Voronoi partitions can be enumerated in $O(n^{(d+2)k+1})$ time.* □

The proof is similar, and is omitted in this version. Combining these results, we have the following.

Theorem 5 *(1) The k-clustering problem for* Var^1 *can be solved in $O(n^{dk+1})$ time.*

(2) The k-clustering problem for Var^2 *can be solved in $O(n^{(d+2)k+1})$ time.*

It should be noted that, from the linear separability or circular separability, of an optimum 2-clustering for

Var^1 and Var^2, respectively, as shown by [2, 7], the k-clustering problem for Var^1 and Var^2 is readily seen to be solvable in $O(n^{dk(k-1)/2})$ and $O(n^{(d+1)k(k-1)/2})$ time, respectively, using similar arguments in [3] for the diameter and radius. Only with the linear/circular separability for the 2-clustering, an algorithm of order $n^{O(dk^2)}$ may be best possible for the k-clustering problem. Our algorithms run in $O(n^{O(dk)})$ time, and improve the $O(n^{O(dk^2)})$ bound greatly. This becomes possible by the fine characterization of optimum k-clusterings by the weighted Voronoi diagram, and by evaluating the primary shatter function of the weighted Voronoi partitions in a tighter manner.

3 Randomized algorithms for the case of the sum of squared errors

The results in the previous section are interesting from the theoretical viewpoint, and the time complexity is polynomial when k is considered as a constant. However, even for $k = 3, 4, 5$, its polynomial degree is quite high, which makes it less interesting to implement the algorithms for practical problems such as the color quantization problem. The k-clustering problem is NP-complete in general when k is regarded as a variable, and in this respect the results are best possible we may expect to have.

To develop a practically useful algorithm, utilizing randomization may be a good candidate, since the intra-cluster metric we are using has its intrinsic statistical meanings. In this section, we develop randomized algorithms for the k-clustering problem with Var^1, the sum of squared error, as the intra-cluster metric.

In this extended abstract, we mainly consider the 2-clustering problem with Var^1, but most of the following discussions carry over to the k-clustering problem. First, let us consider how to estimate $\mathrm{Var}^1(S)$ for the set S of n points $p_i = (\boldsymbol{x}_i)$ $(i = 1, \ldots, n)$ by random sampling. Let T be a set of m points obtained by m independent draws at random from S. If the original point set S are uniformly located, $(n/(m-1))\mathrm{Var}^1(T)$ may be a good estimate for $\mathrm{Var}^1(S)$. However, this is not necessarily the case. For example, suppose that a point p_i in S is far from the other $n-1$ points in S, and the other $n-1$ points are very close to one another. Then, $\mathrm{Var}^1(S)$ is nearly equal to the squared distance between p_i and a point in $S - \{p_i\}$, while with high probability $\mathrm{Var}^1(T)$ is almost zero. This indicates that $\mathrm{Var}^1(T)$ cannot necessarily provide a good estimate for $\mathrm{Var}^1(S)$.

On the other hand, the centroid $\bar{\boldsymbol{x}}(T)$ of T is close to the centroid $\bar{\boldsymbol{x}}(S)$ of S with high probability by the law of large numbers, and we obtain the following lemma.

Lemma 1 *With probability* $1-\delta$*,*

$$\|\bar{\boldsymbol{x}}(T) - \bar{\boldsymbol{x}}(S)\|^2 < \frac{1}{\delta m}\mathrm{Var}^0(S).$$

Proof: First, observe that

$$\mathrm{E}(\bar{\boldsymbol{x}}(T)) = \bar{\boldsymbol{x}}(S), \qquad \mathrm{E}(\|\bar{\boldsymbol{x}}(T) - \bar{\boldsymbol{x}}(S)\|^2) = \frac{1}{m}\mathrm{Var}^0(S)$$

and then apply the Markov inequality to obtain the following.

$$\Pr\Big(\|\bar{\boldsymbol{x}}(T) - \bar{\boldsymbol{x}}(S)\|^2 > \frac{1}{\delta m}\mathrm{Var}^0(S)\Big) < \delta. \quad \square$$

Lemma 2 *With probability* $1-\delta$*,*

$$\sum_{p_i \in S} \|\boldsymbol{x}_i - \bar{\boldsymbol{x}}(T)\|^2 < (1 + \frac{1}{\delta m})\mathrm{Var}^1(S).$$

Proof: Immediate from Lemma 1 and the following.

$$\sum_{p_i \in S} \|\boldsymbol{x}_i - \bar{\boldsymbol{x}}(T)\|^2 = \mathrm{Var}^1(S) + |S| \cdot \|\bar{\boldsymbol{x}}(T) - \bar{\boldsymbol{x}}(S)\|^2. \quad \square$$

Thus, we can estimate $\mathrm{Var}^1(S)$ by random sampling. For the 2-clustering problem, we have to estimate $\mathrm{Var}^1(S_1)$ and $\mathrm{Var}^1(S_2)$ for a 2-clustering (S_1, S_2) by estimating the centroids of S_1 and S_2. Now, consider the following algorithm.

A randomized algorithm for the 2-clustering:

```
1. Sample a subset $T$ of $m$ points from $S$ by $m$ independent draws at random;
2. For every linearly separable 2-clustering $(T_1, T_2)$ of $T$, execute the following:
        Compute the centroids $t_1$ and $t_2$ of $T_1$ and $T_2$, respectively;
        Find a 2-clustering $(S_1, S_2)$ of $S$ by dividing $S$ by the perpendicular bisector of line segment connecting $t_1$ and $t_2$;
        Compute the value of $\mathrm{Var}^1(S_1)+\mathrm{Var}^1(S_2)$ and maintain the minimum among these values;
3. Output the 2-clustering of $S$ with minimum value above.
```

The idea of this randomized algorithm is to use all pairs of centroids of linearly separable 2-clusterings for the sampled point set T. Let (S_1^*, S_2^*) be an optimum 2-clustering of S for Var^1, and let s_1^* and s_2^* be the centroids of S_1^* and S_2^*, respectively. By considering all linearly separable 2-clusterings for T, the algorithm handles the 2-clustering (T_1', T_2') obtained by dividing T by the perpendicular bisector of line segment connecting s_1^* and s_2^*. Then, from the centroids of T_1' and T_2', we obtain a 2-clustering (S_1', S_2') in the algorithm.

Since T is obtained from m independent draws,

$$\mathrm{E}(|T_j'|) = \frac{m}{n}|S_j^*| \quad (j = 1, 2).$$

From Lemma 2, $\mathrm{Var}^1(S_j^*)$ can be estimated by using $|T_j'|$. The sizes $|T_j'|$ $(j = 1, 2)$ are determined by independent Bernoulli trials, and is dependent on the ratio of $|S_1^*|$ and $|S_2^*|$. For the sampling number m, we say that S is $f(m)$*-balanced* if there exists an optimum 2-clustering (S_1^*, S_2^*) with

$$\frac{m}{n}\min\{\,|S_1^*|,\ |S_2^*|\,\} \geq f(m),$$

and the optimum 2-clustering is called an $f(m)$-balanced optimum 2-clustering. We then have the following.

Lemma 3 *Suppose there exists a* $(\log_{\mathrm{e}} m)$*-balanced optimum 2-clustering* (S_1^*, S_2^*)*. Then, with probability* $1 - \dfrac{2}{m^{\beta^2/2}}$ *for a constant* β $(0 < \beta < 1)$*, the following holds.*

$$\min\{\,|T_1^*|,\ |T_2^*|\,\} > (1-\beta)\tfrac{m}{n}\min\{\,|S_1^*|,\ |S_2^*|\,\}$$
$$\geq (1-\beta)\log m.$$

Proof: Set $\mu' = \dfrac{m}{n}\min\{\,|S_1^*|,\ |S_2^*|\,\}$. For m independent Bernoulli trials $X_1, X_2, \ldots, X_m$ with $\Pr(X_i = 1) = \mu'/m \leq \Pr(X_i = 0) = 1 - \mu'/m$, the Chernoff bound implies, for $X = X_1 + \cdots + X_m$,

$$\Pr(X < (1-\beta)\mu') < \exp(-\mu'\beta^2/2).$$

From the assumption,

$$\exp(-\mu'\beta^2/2) \leq \exp(-(\log m)\beta^2/2) = \frac{1}{m^{\beta^2/2}}. \quad \square$$

Theorem 6 *Suppose that the point set* S *is* $f(m)$*-balanced with* $f(m) \geq \log m$*. Then, the randomized algorithm finds a 2-clustering whose total value is within a factor of* $1 + \dfrac{1}{\delta(1-\beta)f(m)}$ *to the optimum value with probability* $1 - \delta - \dfrac{2}{m^{\beta^2/2}}$ *in* $O(nm^d)$ *time.*

Proof: From Lemmas 2 and 3, with probability $1 - \delta - \frac{2}{m^{\beta^2/2}}$,

$$\sum_{j=1}^{2}\sum_{p_i \in S_j'} \|\boldsymbol{x}_i - \bar{\boldsymbol{x}}(T_j')\|^2$$
$$\leq (1 + \tfrac{1}{\delta(1-\beta)f(m)})\sum_{j=1}^{2}\mathrm{Var}^1(S_j^*)$$

holds. Furthermore, the left hand side is bounded from below by $\sum_{j=1}^{2}\mathrm{Var}^1(S_j')$, whose value is computed in the algorithm. Hence, the minimum value found in the algorithm is within the factor.

Concerning the time complexity, all linearly separable 2-clusterings for T can be enumerated in $O(m^d)$ time. For each 2-clustering (T_1, T_2) of T, finding a pair of centroids and a 2-clustering of S generated by the pair together with its objective function value can be done in $O(n)$ time. Thus the theorem follows. $\square$

We have developed a randomized algorithm only for the 2-clustering problem so far, but this can be directly generalized to the k-clustering problem. If there exists a balanced optimum k-clustering, similar bounds can be obtained. It may be noted that the technique employed here has some connection with the technique used to obtain a deterministic approximate algorithm with worst-case ratio bounded by 2 for the k-clustering problem in [7].

The above theorem assumes some balancing condition. In some applications, a very small cluster is useless even if its intra-cluster is small. For example, when we apply a 2-clustering algorithm recursively in a top-down fashion to solve the k-clustering problem, a balancing condition on 2-clusterings may be imposed to 2-clustering subproblems so that the sizes of subproblems may become small quickly and the total clustering may have nicer properties. In such a case, the randomized algorithm naturally ignores such small-size cluster. Also, for the case of finding a good and balanced 2-clustering, we have only to apply a slightly modified version of the randomized algorithm directly. This is typical for the clustering problem in VLSI layout design. See, for example, Kernighan and Lin [13]. Generalizing Theorem 6 for such cases, we have the following.

Theorem 7 *For the problem of finding an optimum 2-clustering among* (γm)*-balanced 2-clusterings for a constant* γ*, the randomized algorithm finds a 2-clustering which is almost at least* (γm)*-balanced and whose value is within a factor of* $1+O(1/(\delta m))$ *to the optimum value of this problem with probability* $1 - \delta$ *for small* δ*.* $\square$

In the proof of this theorem, we use results concerning the ϵ-net and ϵ-approximations. The details will be given in the full version. We here restate the theorem in a rough way so that its essence can be understood direclty.

Corollary 1 *When there is an optimum 2-clustering (S_1^*, S_2^*) of S such that*

$$|S_1^*|,\ |S_2^*| \geq \gamma |S| = n$$

for some constant γ $(0 < \gamma \leq 1/2)$, the randomized algorithm, which samples m points from S, finds a 2-clustering whose expected value is $1 + O(1/m)$ with high probability. □

Corollary 2 *For the problem of finding an optimum 2-clustering among 2-clusterings (S_1, S_2) of S such that*

$$|S_1|,\ |S_2| \geq \gamma |S| = n$$

for some constant γ $(0 < \gamma \leq 1/2)$, the randomized algorithm, which samples m points from S, can be easily modified to find an approximate balanced 2-clustering whose expected value is $1+O(1/m)$ with high probability. □

On the other hand, if very small clusters with small intra-cluster metric should be found, we may enumerate such small clusters deterministically or in a randomized manner, since the number of such small clusters is relatively small. For the 2-clustering problem in the two-dimensional case, the number of linearly separable 2-clustering such that one cluster consists of at most k' points is $O(k'n)$ and can be enumerated efficiently. By enumerating k'-sets for an appropriate value of k', we obtain the following theorem.

Theorem 8 *The 2-clustering problem for n points in the plane with* Var^1 *as the intra-cluster metric can be solved in $O(n^{5/3}(\log n)^3)$ time with the approximation ratio within a factor of $1 + O(1/\log n)$ with probability $1 - O(1/\log n)$.*

Proof: Set $m = n^{1/3} \log n$. For this m, by the randomized algorithm, we find a good $(\log m)^2$-balanced 2-clustering. Also, by the deterministic algorithm enumerating $(\leq n^{2/3} \log n)$-sets, we find a best unbalanced 2-clustering. Setting $\delta = 1/\log m$ and β to a constant, the time complexity of the randomized algorithm is $O(n(n^{1/3} \log n)^2) = O(n^{5/3}(\log n)^2)$ and the approximation ratio is bounded by $1 + O(1/\log m)$ with probability $1 - O(1/\log m)$. The deterministic algorithm runs in $O(n(n^{2/3} \log n)(\log n)^2) = O(n^{5/3}(\log n)^3)$ time. □

It should be noted that the time complexity in this theorem is subquadratic, compared with the deterministic quadratic exact algorithm.

4 Concluding remarks

We have demonstrated that optimum solutions to the variance-based k-clustering can be characterized by the (weighted) Voronoi diagram generated by k points, and have evaluated the primary shutter function of the k-Voronoi space. This primary shutter function can be used in computational learning theory in learning k-Voronoi spaces.

We have then presented a simple randomized algorithm for the k-clustering problem with Var^1 as an intra-cluster metric. This algorithm is practically useful when k is small and balanced k-clusterings are preferable. For example, for the problem of finding an optimum 2-clustering for n planar points among almost completely balanced 2-clusterings, an approximate 2-clustering which is approximately balanced and whose cost is within a factor of $1+1/3 = 4/3$ on the average to the optimum cost can be found by sampling 10 points from n points and spending $O(10^2 n) = O(n)$ time with probability $\sum_{i=3}^{7} \binom{10}{i}/2^{10} \approx 0.89$, or by sampling 20 points and spending $O(20^2 n) = O(n)$ time with probability $\sum_{i=3}^{17} \binom{20}{i}/2^{20} \approx 0.9996$.

The randomized algorithm, however, is not so suitable to find a good unbalanced 2-clustering. Also, although the randomized algorithm itself is valid for large k, the running time becomes inherently large since the primary shutter function for m sampled points is $O(m^{O(dk)})$. This implies that the variance-based k-clustering problem is harder in such a situation compared with the diameter- or radius-based k-clustering to which a good and fast approximate algorithm with worst-case ratio 2 is known [4].

Therefore, to solve the variance-based k-clustering for large k practically, say for $k = 256$ of the typical color quantization problem, we may apply the randomized 2-clustering algorithm proposed in this paper recursively in a top-down manner with sampling only a small number of points at each stage as mentioned above. We plan to implement this approach and experimentally evaluate its efficiency.

Acknowledgment

The work of the authors was supported in part by the Grant-in-Aid of Ministry of Education, Science and Culture of Japan.

References

[1] T. Asano, B. Bhattacharya, M. Keil and F. Yao: Clustering algorithms based on minimum and max-

imum spanning trees. *Proceedings of the 4th Annual Symposium on Computational Geometry*, Urbana, 1988, pp.252–257.

[2] E. Boros and P. L. Hammer: On clustering problems with connected optima in Euclidean spaces. *Discrete Mathematics*, Vol.75 (1989), pp.81–88.

[3] V. Capoyleas, G. Rote and G. Woeginger: Geometric clustering. *Journal of Algorithms*, Vol.12 (1991), pp.341–356.

[4] T. Feder and D. H. Greene: Optimal Algorithms for Approximate Clustering. *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, 1988, pp.434–444.

[5] J. Hagauer and G. Rote: Three-clustering of points in the plane. *Proceedings of the 1st Annual European Symposium on Algorithms (ESA'93)*, Lecture Notes in Computer Science, Vol.726, 1993, pp.192–199.

[6] S. Hasegawa: *A study on ϵ-net and ϵ-approximation.* Master's Thesis, Department of Information Science, University of Tokyo, 1993.

[7] S. Hasegawa, H. Imai, M. Inaba, N. Katoh and J. Nakano: Efficient algorithms for variance-based k-clustering. *Proceedings of the First Pacific Conference on Computer Graphics and Applications*, World Scientific, 1993, pp.75–89.

[8] S. Hasegawa, H. Imai and M. Ishiguro: ϵ-approximations of k-label spaces. *Proceedings of the 4th International Workshop on Algorithmic Learning Theory (ALT'93)*, Lecture Notes in Artificial Intelligence, Vol. 744, 1993, pp.288–299.

[9] P. Heckbert: Color image quantization frame buffer display. *ACM Transactions on Computer Graphics*, Vol.16, No.3 (1982), pp.297–304.

[10] J. Hershberger: Minimizing the sum of diameters efficiently. *Computational Geometry: Theory and Applications,* Vol.2 (1992), pp.111-118.

[11] M. Ishiguro: *Evaluation of combinatorial complexity for hypothesis spaces in learning theory with applications.* Master's Thesis, Department of Information Science, University of Tokyo, 1994.

[12] N. Katoh and T. Ibaraki: A parametric characterization and an ϵ-approximation scheme for the minimization of a quasiconcave program. *Discrete Applied Mathematics*, Vol.17 (1987), pp.39–66.

[13] B. W. Kernighan and S. Lin: An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, Vol.49, No.2 (1970), pp.291–307.

[14] N. Megiddo and K. J. Supowit: On the complexity of some common geometric location problems. *SIAM Journal on Computing*, Vol.13 (1984), pp.182–196.

[15] S. J. Wan, S. K. M. Wong and P. Prusinkiewicz: An algorithm for multidimensional data clustering. *ACM Transactions on Mathematical Software*, Vol.14, No.2 (1988), pp.153–162.

Bounded boxes, Hausdorff distance, and a new proof of an interesting Helly-type theorem.

Nina Amenta [*]
The Geometry Center
1300 South Second Street
Minneapolis, MN. 55454

Abstract

In the first part of this paper, we reduce two geometric optimization problems to convex programming: finding the largest axis-aligned box in the intersection of a family of convex sets, and finding the translation and scaling that minimizes the Hausdorff distance between two polytopes. These reductions imply that important cases of these problems can be solved in expected linear time. In the second part of the paper, we use convex programming to give a new, short proof of an interesting Helly-type theorem, first conjectured by Grünbaum and Motzkin.

1 Introduction

Linear programming is a popular tool in computational geometry. A related problem is

Convex Programming
Input: *A finite family H of closed convex sets in E^d and a convex function f.*
Output: *The minimum of f over $\bigcap H$, the intersection of H.*

Remember that a function $f : E^d \rightarrow \mathcal{R}$ is convex when $f(\lambda a + (1-\lambda)b) \leq \lambda f(a) + (1-\lambda)f(b)$, for all $a, b \in E^d$ and $0 \leq \lambda \leq 1$.

The elements of the family H are called the *constraints.* Like a linear program, a convex program has a single global minimum which is determined by a subfamily of constraints. And under reasonable computational assumptions on the constraints and the convex function, fixed-dimensional convex programming can be solved in expected linear (in $|H|$) time by any of the randomized combinatorial linear programming algorithms of [C90], [S90], [MSW92]. These algorithms are combinatorial in the sense that they operate by searching the subfamilies of constraints for one that defines the minimum; a more classic example of a combinatorial LP algorithm the simplex algorithm.

We give two new reductions from geometric optimization problems of practical interest to convex programming. The first problem is finding the largest volume axis-aligned box in the intersection of a family K of convex sets. Like the ubiquitous bounding box, this *bounded box* can be used in heuristics to approximate a more complicated volume. When the dimension is fixed and the elements of K are linear halfspaces (or otherwise easy to compute with), this reduction implies an expected $O(n)$ time algorithm, $n = |K|$. The planar case of this problem was raised in the context of a heuristic for packing clothing pattern pieces [DMR93]. They give an $O(n\alpha(n)\log n)$ algorithm to find the maximum area axis-aligned rectangle in any simple polygon in E^2. Our reduction may be useful for higher dimensional packing heuristics as well, and also when objects are to be decomposed into collections of axis-aligned boxes (eg. in ray tracing, [AK89],pp 219-23).

Our second reduction concerns some particular cases of the much-studied problem of minimizing the Hausdorff distance between two ob-

[*] email: nina@geom.umn.edu. Most of this work was done at the University of California, Berkeley, where the author was supported by a U.C. Presidents Dissertation Year Fellowship and an AT&T Graduate Research Program for Women Grant.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

jects under a group of transformations. The *one-directional Hausdorff distance* from a set A to a set B, $\vec{H}(A,B)$, is the maximum distance from any point in A to the nearest point in B. The *Hausdorff distance* between A and B, $H(A,B)$, is $\max\{\vec{H}(A,B), \vec{H}(B,A)\}$. The Hausdorff distance is used in pattern recognition and computer vision as a a measure of the difference in shape between the two sets; typically a family P of critical points is extracted from an image and compared with a stored template A. The problem is to choose a transformation from some family which, when applied to A, minimizes $H(P,A)$ or $\vec{H}(P,A)$.

For convex polytopes A, B we reduce the problem of finding the translation and scaling which minimizes $H(A,B)$ to a convex program. In the plane, this convex program can be solved in expected $O(n)$ time, where n is the total number of vertices. A similar convex program, motivated by the scenario above, finds the translation which minimizes the $\vec{H}(P,A)$, where P is a point set and A is a convex polytope. Again in the planar case, this convex program requires expected $min\{O(mn), O(n \lg n + m)\})$ time, where $n = |P|$ and A has m vertices.

There are many results on minimizing the Hausdorff distance between various objects under different groups of motions. Most of them solve more difficult problems and require more time. For planar point sets P, Q, an algorithm of [HKS91] finds the translation that minimizes $H(P,Q)$ in $O(mn(m+n)\lg(mn))$ time, where $n = |P|, m = |Q|$. Allowing rotation as well, an algorithm of [CGHKKK94] minimizes $H(P,Q)$ in $O(m^2n^2(m+n)\lg^2 mn)$ time. Algorithms for measuring the Hausdorff distance for fixed polygons, allowing no transformations at all, require $O(n \lg n)$ time for simple polygons [ABB91], and $O(n)$ for convex polygons [A83]. The algorithms implied by our reductions are also comparatively simple and implementable.

We turn from these practical issues to a theoretical question from combinatorial geometry. We give a new proof of the theorem of [Mo73]:

Theorem 1.1 *Let C_d^k be the family of all sets in R^d consisting of the disjoint union of at most k closed convex sets. Let $I_d^k \subset C_d^k$ be a subfamily with the special property that it is closed under intersection. Then the intersection of I_d^k is nonempty if and only if the intersection of J is nonempty, for all $J \subseteq I_d^k$ with $|J| \leq k(d+1)$.*

This is called a *Helly-type theorem* because of its combinatorial structure: the whole family of sets has nonempty intersection if and only if all of its constant-size subfamilies do. There are many Helly-type theorems; it is appealing to think that there is some fundamental topological property underlying them all. This theorem is interesting because it suggests this fundamental property might be that the intersection of every subfamily is somehow homologically of constant complexity.

Grünbaum and Motzkin conjectured Theorem 0.1 [GM61], and proved the case $k = 2$, using a more general axiomatic structure in place of convexity. The case $k = 3$ was proved by Larman [L68]. Morris settled the conjecture in his thesis. His proof, however, is quite long (69 pages) and involved, and a better proof has been called for [E93].

Using convex programming, we give a short and insightful proof. Our approach is to introduce a function f, and then show that the problem of minimizing f over a family I_d^k belongs to the class *GLP* (for Generalized Linear Programming). Informally, GLP is the class of problems which can be solved by combinatorial LP algorithms. This is already interesting, as an example of a problem which is combinatorially similar to LP although geometrically the intersection of the constraints fails not only to be convex, but even to be connected. Theorem 0.1 follows by applying an easy theorem from [A93], that there is a Helly-type theorem about the constraint set of every GLP problem.

2 Setup

In this section we give some background on convex programming, GLP, and the combinatorial LP algorithms.

First we give the formal definition of GLP, using an abstract framework due to Sharir and Welzl [SW92]. A GLP problem is a pair (H, w), where H is a family of constraints, generally sets, and $w : 2^H \rightarrow \Lambda$ is a function which takes a subfamily of constraints to an element of a totally ordered set Λ. Λ has a special maximum element Ω. When

$w(G) < \Omega$, G is *feasible*, and otherwise G is *infeasible.*

In convex programming, for example, $w(G)$, for $G \subseteq H$, is the minimum of f over $\bigcap G$, and if $\bigcap G = \emptyset$, then $w(G) = \Omega$. In order to ensure that w is defined on every subfamily of constraints, we may have to enclose the problem in a compact "bounding box". We say that w is the objective function *induced* by f.

(H, w) is a GLP problem if

1. For all $F \subseteq G \subseteq H$: $w(F) \leq w(G)$

2. For all $F \subseteq G \subseteq H$ such that $w(F) = w(G)$ and for each $h \in H$:
$w(F+h) > w(F)$ if and only if $w(G+h) > w(G)$ (by $F+h$, we mean $F \cup \{h\}$)

A *basis* is a subfamily $G \subseteq H$ such that $w(G-h) < w(G)$, for all $h \in G$. The *combinatorial dimension* of a GLP is the maximum cardinality of any feasible basis.

Every convex program meets Condition 1, since adding more constraints to a problem can only increase the minimum. It is also well-known that a d-dimensional convex program has combinatorial dimension d; there is a formal proof in [A93'].

But notice that it is not the case that every convex program is a GLP problem, since it may fail to satisfy Condition 2. Condition 2 is always satisfied, however, when the minimum of f over the intersection of every subfamily $G \subseteq H$ is achieved by exactly one point. This observation implies that the following cases of convex programming are always GLP problems of combinatorial dimension d.

A function $f : E^d \rightarrow \mathcal{R}$ is *strictly convex* when $f(\lambda a + (1-\lambda)b) < \lambda f(a) + (1-\lambda)f(b)$, for all $a, b \in E^d$ and $0 < \lambda < 1$.

Strictly convex programming
Input: *A family H of compact convex subsets of E^d, and a strictly convex function f.*
Output: *The minimum of f over $\bigcap H$.*

The minimum is achieved at one unique point since any point x on the the line segment between any two feasible points y, z with $f(y) = f(z)$ has $f(x) < f(y) = f(z)$.

The *lexicographic function* $f : E^d \rightarrow \mathcal{R}^d$ takes a point to it's coordinates $x =< x_1, \ldots, x_d >$, which are totally ordered so that $x > y$ if $x_1 > y_1$, or if $x_1 = y_1$ and $x_2 > y_2$, and so on.

Lexicographic convex programming
Input: *A family H of compact convex subsets of E^d, and the lexicographic function f.*
Output: *The minimum of f over $\bigcap H$.*

The minimum is certainly achieved at a single point, since each point has a unique value. But this is not, strictly speaking, a convex programming problem, since f is not a function into $\mathcal{R}$. We observe, however, that f is related to the linear function

$$g_\epsilon(x) = x_1 + \epsilon x_2 + \epsilon^2 x_3 + \ldots + \epsilon^{d-1} x_d$$

for infinitessimally small ϵ. To make this relationship precise, we adopt the terminology of [M94], and say that an objective function v is a *refinement* of a function w when, for $F, G \subseteq H$, $w(G) > w(F)$ implies $v(G) > v(F)$.

Let w and v_ϵ be the objective functions induced by f and g_ϵ, respectively. For any finite family H of constraints, there is some ϵ small enough so that v_ϵ is a refinement of w. Note that there may be no ϵ small enough that $v_\epsilon(F) = v_\epsilon(G)$ whenever $w(F) = w(G)$, as illustrated in figure ??. Finally, we observe that if a function w has some refinement v such that (H, v) meets Condition 1 and has combinatorial dimension d, then so does (H, w). So Lexicographic Convex Programming is a GLP problem of combinatorial dimension d, since (H, v_ϵ) is a convex program and so meets Condition 1 and has combinatorial dimension d, and (H, w) meets Condition 2 as well.

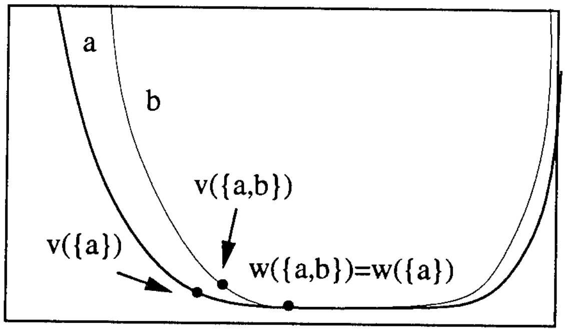


Figure 1: Minima are different under v_ϵ

The computational requirement under which any of the combinatorial LP algorithms can be applied to any GLP problem is that there is a subroutine available for the following problem:

Basis computation

Input: *A basis G and a constraint h.*
Output: *A basis $G' \subseteq G + h$ such that $w(G') = w(G+h)$.*

This operation corresponds to a pivot step in the simplex algorithm. In d-dimensional convex programming, a basis computation minimizes f over $\bigcap G$, where $|G| \leq d+1$. When a basis computation can be done in constant time, then any of [C90], [S90], [MSW92] require expected $O(n)$ time, where $n = |H|$. All of these algorithms have been implemented for LP, and the algorithm in [S90] has also been applied to the particular convex program of finding the smallest ball enclosing a family of points in E^d [W91].

3 Bounded boxes

In this section we prove

Theorem 3.1 *Finding the largest volume axis-aligned box in the intersection of a family K of n convex bodies in E^d is a strictly convex program in E^{2d} with $2^d n$ constraints.*

Proof sketch: We parameterize an axis-aligned box by a pair of vectors $x, a \in \mathcal{R}^d$, where $x_1, \ldots, x_d$ are the coefficients of the lexicographically minimum vertex of the box, and $a_1, \ldots, a_d$ are positive offsets in each coordinate direction. This parameteriztion defines a space of boxes, in which we will construct a convex program.

For each convex body $C \in K$, we will define 2^d constraints, one for each box vertex. Note that a box is contained in a convex body C if and only if all of its vertices are. Let us label the vertices with 0-1 vectors in the natural way, so that $(0, \ldots, 0)$ is the lexicographic minimum corner of the box. The set of boxes for which vertex u is contained in C is $h = \{x, a \mid x + (u \otimes a) \in C\}$ (here $\otimes$ is coordinatewise multiplication, and $+$ is translation). This is convex.

To prevent the largest volume box determined any subproblem from being unbounded, we require the bounded box to be contained in a very large bounding box which is guranteed to contain $\bigcap K$. This adds one more convex constraint to the problem in the space of boxes.

It remains to show that maximizing the volume of the box corresponds to minimizing some convex function over $\bigcap H$. The volume of a box (x, a), negated, is given by

$$g(a) = -\prod_{i=1}^{d} a_i$$

with all the a_i constrained to be positive. This is not a convex function, but

$$f(a) = -\log(\prod_{i=1}^{d} a_i) = -\sum_{i=1}^{d} \log(a_i)$$

is a strictly convex function. Minimizing f over $\bigcap H$ is a strictly convex programming problem in E^{2d}.

□

When the elements of K are linear halfspaces, then the constraints are as well. When the elements of K are of constant complexity, so that a basis computation requires $O(1)$ time for fixed d, the largest volume axis-aligned box can be found in expected $O(n)$ time. Note that f and g are minimized at the same point, so the basis computation may be implemented using g.

4 Hausdorff distance

Now we consider the problem of finding the translation and scaling which minimize the Hausdorff distance between two convex polytopes A and B in E^d. The boundary of the set of points at distance λ from A (that is, the Minkowski sum of A with the closed disk of radius λ centered at the origin) is called the λ-*offset surface.* $\vec{H}(B, A) \leq \lambda$ when every vertex of B lies within the λ-offset surface of A.

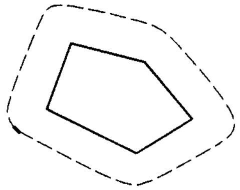


Figure 2: Offset surface

We can think of the scaling and translation transformations as being applied to A alone. We define a $(d+2)$-dimensional *transformation space* in which the coordinates of each point represent a

d-dimensional translation vector τ to be applied to A, a scale factor σ to be applied to A, and an offset distance λ. For any point $b \in B$, let $H^b_{(\tau,\sigma,\lambda)}$ be the subset of transformation space such that b is within the λ-offset surface of the homothet of A scaled by σ and translated by τ. Similarly, let $H^a_{(\tau,\sigma,\lambda)}$ be the set of translations and scalings of A which put a point $a \in A$ inside the λ-offset surface of B.

Theorem 4.1 *The scaling and translation that minimizes the Hausdorff distance between two polytopes in E^d can be found by a lexicographic convex program in E^{d+2}.*

Proof: Let $V(A)$ and $V(B)$ be the vertex sets of A and B, respectively. Consider a vertex $b \in V(B)$. Fixing $\sigma = 1$ and $\lambda = 0$, we find that the set of translations of A which cover b, $H^b_{(\tau,1,0)}$, is itself a translate of the convex set $-A$. Allowing σ to vary, we find that $H^b_{(\tau,\sigma,0)}$ is a cone over $-A$, also convex, into the direction of the σ coordinate. Finally, we allow λ to vary as well. Notice that, as λ varies, the set of disks of radius λ centered at the origin forms a convex cone, C. $H^b_{(\tau,\sigma,\lambda)}$ is the Minkowski sum of $H^b_{(\tau,\sigma,0)}$ with C. This is convex because the Minkowski sum of convex bodies is convex. Each vertex of B produces one such convex constraint.

Now consider a vertex $a \in V(A)$. Again $H^a_{(\tau,1,0)}$ is convex, this time a translate of $-B$. As σ varies, A scales, and vertex a moves in some direction v, along line $a + \sigma v$. So $H^a_{(\tau,\sigma,0)}$ is a convex cylinder over $-B$, and $H^a_{(\tau,\sigma,\lambda)}$ is the Minkowski sum of the cylinder with C.

We use the objective function given by the lexicographic function f on the transformation space, where λ is the most significant coordinate, followed by σ and then the d coordinates of the translation τ. The minimum point in the intersection of the constraints $H^a_{(\tau,\sigma,\lambda)}$ and $H^b_{(\tau,\sigma,\lambda)}$, with respect to f, represents a scaling and translation which minimizes the Hausdorff distance between A and B.
□

In the plane, the most important case for existing applications, we can implement this convex program so that

Theorem 4.2 *The scaling and translation which minimizes the Hausdorff distance between two polygons in the plane can be found in expected $O(n)$ time, where $n = |V(A)| + |V(B)|$.*

Proof: We redefine each polygon as the intersection of pieces of constant complexity, and associate each piece with a subset of the vertices of the other polygon. Each piece is the infinite wedge formed by a vertex of one of the polygons and the rays supporting the adjacent sides, which we shall call an *angle.* If every vertex of A is within the λ-offset surface of every angle from B, and visa versa, $H(A, B)$ is no greater than λ. This gives a GLP with $O(n^2)$ constraints, pairing every vertex of A with every angle from B, and visa versa.

We get a linear number of constraints by noting that for every angle α from B, all of A is within the λ-offset surface of α if every vertex from a *critical subset* of $V(A)$ is. The faces of B divide the circle of normal directions into a family I_B of closed intervals, each interval corresponding to an angle. A vertex v of A is critical for an angle α if v is extremal in A for any direction in the interval induced by α. The face normals of A also divide the circle of normal directions into a family I_A of closed intervals, each corresponding to the set of directions in which a particular vertex of A is extremal. When the interval corresponding to a vertex v intersects the interval corresponding to an angle α, v is extremal for α. The critical subsets for every angle can be found by merging I_A and I_B in linear time. Each of the n intersections of an interval in I_A with an interval in I_B gives a vertex-angle pair which produces a constraint. We also construct the n constraints induced by angles of A and vertices of B.

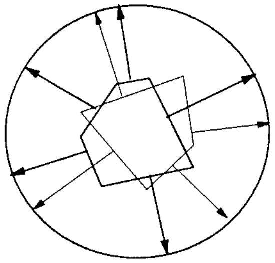


Figure 3: Intervals on the circle of normals

The lexicographic minimum point in the intersection of any four constraints can be found in constant time, so we get an expected $O(n)$ time algorithm.
□

We now turn our attention to the problem of minimizing the one-directional Hausdorff distance $\vec{H}(P, A)$ from a set of P points to a convex polytope A. We can always scale A so that it is large enough to cover all the points, making $\vec{H}(P, A)$ zero. When the only transformation allowed is translation, however, we find

Theorem 4.3 *The translation that minimizes the one-directional Hausdorff distance from a set P of points to a convex polytope A in E^d can be found by a lexicographic convex program in E^{d+1}. For $d = 2$, we can solve the problem in expected $min\{O(mn), O(n \lg n + m)\}$ time, where $n = |P|$ and $m = |V(A)|$.*

Proof sketch: If the $O(mn)$ term is less than the $O(n \lg n + m)$ term (few vertices and many points), we construct a convex program in which each point in P produces n constraints of constant complexity, similar to those in the previous reduction. If not, we compute the convex hull of P and use a one-directional, translation-only modification of the previous construction.

Note that the Hausdorff distance used in this section can be derived from any metric on E^d, not just L^2.

5 A new proof of an interesting Helly-type theorem

In this section we will use an easy but powerful theorem from [A93], which gives us a simple technique for proving Helly-type theorems.

Theorem 5.1 *Let (H, w) be a GLP problem with combinatorial dimension d. The intersection of H is nonempty if and only if the intersection of every $G \subseteq H$ with $|G| \leq d+1$ is nonempty.*

Let C_d^k be the family of all sets in R^d consisting of the disjoint union of at most k closed convex sets. A family I of sets is *intersectional* if, for every $H \subseteq I$, $\bigcap H \in I$. C_d^k is not intersectional. But consider some subfamily $I_d^k \subset C_d^k$ which *is* intersectional. This may "just happen" to be true, or I_k^d may be intersectional for some geometric reason. For example consider of any family J of sets like the one in figure 4 where each set is a pair of balls of diameter δ, separated by a distance of at least δ, kind of like dumbbells. The family formed by taking the intersection of every subfamily of J forms an intersectional family I_d^k, since every intersection consists of at most two convex components.

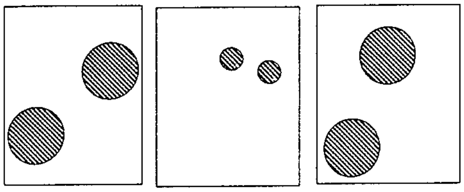


Figure 4: Generating family of I_d^k

Morris [Mo73] proved the following

Theorem 5.2 *Any intersectional family $I_d^k \subseteq C_d^k$ has Helly number $k(d+1)$.*

We use Theorem 4.1 to give a new short and intuitive proof. Given any finite family $H \subseteq I_d^k$, we construct a GLP problem with H as the constraints. Let f be the lexicographic objective function on E^d, and define $w(G) = \min\{f(x) \mid x \in \bigcap G\}$, for all $G \subseteq H$. Since minima are identified with points, we will speak of $w(G)$ as a point and ignore the fine distinction between the point x itself and the value $f(x)$.

Theorem 5.3 *Finding $w(H)$ is a GLP problem of combinatorial dimension $k(d+1) - 1$.*

Proof: It is easy to see that the problem satisfies Condition 1, since adding constraints to a subproblem can only increase the minimum. And is satisfies Condition 2 since every value of $w(G)$ is identified with a unique point, which is either in or out of h.

Recall that the combinatorial dimension is the largest cardinality of any basis B such that $\bigcap B$ is nonempty. We will count the constraints in any basis B by carefully removing selected constraints one by one, while building up a subfamily S of "sacred" constraints which may not be removed in later steps.

We will maintain two invariants. The first is that $w(B - h) < w(B)$ for all $h \in B - S$. The second invariant is that for all $h \in B - S$, the minimum point $w(B - h)$ lies in a different convex component of $\bigcap(B - h)$ from the point $w(B)$.

First we choose the subfamily S so that the invariants are true initially. Since $\bigcap B \neq \emptyset$, there is a minimum point $w(B)$ in some convex component of $\bigcap B$. Each $h \in B$ is the disjoint union of convex sets; for each h, the point $w(B)$ is contained in exactly one of them. Call this convex set C_h, and let $C = \{C_h \mid h \in B\}$. The pair (C, w) is a lexicographic convex programming problem, a GLP problem of combinatorial dimension d, with $w(C) = w(B)$. So C must contain a basis B_C with $|B_C| \leq d$. We set $S = \{h \in B \mid C_h \in B_C\}$.

How does this ensure the invariants? Since B is a basis, the first invariant holds for any subset S. The second invariant holds because all the constraints which contributed a convex component to B_C are in S, and for any $h \in B - S$, $w(B - h) < w(B) = w(B_C)$. That is, since the point $w(B)$ is the lowest point in $\bigcap B_C$, and $w(B - h)$ is lower than $w(B)$, the point $w(B-h)$ cannot be in $\bigcap B_C$, and hence must be in a different convex component of $\bigcap(B - h)$.

Now we turn our attention to selecting a constraint to remove from B. We use the fact that all the points $w(B-h)$ are distinct, for all $h \in B-S$. This is true because the point $w(B - h) \notin h$, so that for any other $h' \in B$, since $h \in (B - h')$, $w(B - h) \notin \bigcap(B - h')$. Since the $w(B - h)$ are distinct, there is some $h_{max} \in B - S$ such that $w(B - h_{max}) > w(B - h)$ for all other $h \in B - S$.

So consider removing h_{max} from B. Since $w(B - h) < w(B - h_{max})$, for any other $h \in B - S$, certainly $w(B - h - h_{max}) < w(B - h_{max})$. So the first invariant is maintained for $B - h_{max}$ and S. To re-establish the second invariant, we have to add more elements to S. We do this in the same way as before, by finding the at most d constraints which determine the minimum of the convex component containing $w(B - h_{max})$. We add these constraints to S, and set $B = B - h_{max}$.

We iterate this process, selecting constraints to remove from B and adding constraints to S, until $B - S$ is empty, that is, $B = S$. We now show that each removed constraint h accounts for at least one convex component C_h in $\bigcap S$. Removing h from B caused a new minimum point $w(B - h)$ to be created. This point was the minimum point in some convex component C_h of $\bigcap(B - h)$. We added the constraints determining $w(B - h)$ to S, so $w(B-h)$ has to remain the minimum point in its convex component, throughout the rest of the process. This ensures that, although C_h may later become part of some larger component, it will never become part of a larger component with a lower minimum point. Each subsequent component created by the removal of another constraint from B *will* in fact have a lower minimum point, so the component containing $w(B - h)$ must remain distinct from all later components. Thus every removed constraint h will account for at least one distinct component in $\bigcap S$.

Since I_k^d is an intersectional family, no subfamily of constraints can have more than k convex components in its intersection. Since $\bigcap B$ was initially nonempty, we started with at least one convex component, and at most d constraints in S. No more than $k-1$ constraints were removed, and each constraint removed added at most d constraints to S. So the total size of $|B| \leq (k-1)+|S| \leq k(d+1)-1$.

□

Theorems 4.1 and 4.3 together imply Theorem 4.2.

References

[A93] Nina Amenta, Helly Theorems and Generalized Linear Programming, *Proceedings of the 7th Annual Symposium on Computational Geometry*, (1993) pages 63-72.

[A93'] Nina Amenta, *Helly Theorems and Generalized Linear Programming* (PhD thesis), Geometry Center Technical Report, Number GCG61, (1993).

[ABB91] Helmut Alt, Bernd Behrends, and Johannes Blömer. Approximate matching of polygonal shapes, *Proceedings of the 7th Annual Symposium on Computational Geometry*, (1991), pages 186-93.

[AK89] James Arvo and David Kirk. A survey of ray tracing acceleration techniques, in *An Introduction to Ray Tracing* (Andrew Glassner, ed), Academic Press, San Diego, (1989), pages 201-262.

[A83] Mikhail J. Atallah. A linear time algorithm for the Hausdorff distance between convex polygons, *Information Processing Letters* **17** (1983), pages 207-9.

[CGHKKK94] L. Paul Chew, Michael T. Goodrich, Daniel P. Huttenlocher, Klara Kedem, Jon M. Kleinberg, and Dina Kravets. Geometric Pattern Matching under Euclidean Motion, *Proceedings of the Fifth Canadian Conference on Computational Geometry*, (1993) pages 151-156

[C90] Kenneth L. Clarkson. Las Vegas algorithms for linear and integer programming when the dimension is small, *manuscript*, 1990. An earlier version appeared in *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 452-5, 1988.

[DMR93] Karen Daniels, Victor Milenkovic and Dan Roth. Finding the maximum area axis-parallel rectangle in a polygon, *Proceedings of the Fifth Canadian Conference on Computational Geometry*, (1993) pages 322-327.

[DGK63] Ludwig Danzer, Branko Grünbaum, and Victor Klee. Helly's Theorem and it's relatives, *Proceedings of the Symposium on Pure Mathematics*, Vol. 7, Convexity (1963) pages 101-180. American Mathematical Society, Providence, RI.

[E93] Jürgen Eckhoff. Helly, Radon and Carathody type theorems, Chapter 2.1 in *Handbook of Convex Geometry*, P.M. Gruber and J.M. Willis, eds., (1993) Elsevier Science Publishers B. V., Amsterdam.

[GM61] Branko Grünbaum and Theodore S. Motzkin. On components in some families of sets, *Proceedings of the American Mathematical Society*, vol. 12, (1961) pages 607-613.

[HKS91] Daniel P. Huttenlocher, Klara Kedam and Micha Sharir. The Upper Envelope of Voronoi Surfaces and its Applications, *Proceedings of the 7th Annual Symposium on Computational Geometry*, (1991) pages 194-203.

[L68] D.G. Larman. Helly type properties of unions of convex sets, *Mathematika* 15 (1968) pages 53-59.

[M94] Jiří Matoušek. On geometric optimization with few violated constraints, *These Proceedings.*

[MSW92] Jiří Matoušek, Micha Sharir and Emo Welzl. A subexponential bound for linear programming, *Proceedings of the 8th Annual Symposium on Computational Geometry* (1992) pages 1-8.

[Mo73] Howard Cary Morris. *Two Pigeon Hole Principles and Unions of Convexly Disjoint Sets*, PhD thesis, California Institute of Technology, Pasadena, California (1973).

[S90] Raimund Seidel. Linear programming and convex hulls made easy, *Proceedings of the 6th Annual Symposium on Computational Geometry*, (1990) pages 211-215.

[SW92] Micha Sharir and Emo Welzl. A combinatorial bound for linear programming and related problems, *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science* (1992), Lecture Notes in Computer Science 577, pages 569-579.

[W91] Emo Welzl. *Smallest enclosing disks (balls and ellipsoids)*, Technical report number B 91-09 Fachbereich Mathematik, Freie Universität Berlin, Berlin, Germany (1991)

Computing Envelopes in Four Dimensions with Applications[*]

Pankaj K. Agarwal[†] Boris Aronov[‡] Micha Sharir[§]

Abstract

Let $\mathcal{F}$ be a collection of n d-variate, possibly partially defined, functions, all algebraic of some constant maximum degree. We present a randomized algorithm that computes the vertices, edges, and 2-faces of the lower envelope (i.e., pointwise minimum) of $\mathcal{F}$ in expected time $O(n^{d+\varepsilon})$, for any $\varepsilon > 0$. For $d = 3$, by combining this algorithm with the point location technique of Preparata and Tamassia, we can compute, in randomized expected time $O(n^{3+\varepsilon})$, for any $\varepsilon > 0$, a data structure of size $O(n^{3+\varepsilon})$ that, given any query point q, can determine in $O(\log^2 n)$ time whether q lies above, below or on the envelope. As a consequence, we obtain improved algorithmic solutions to many problems in computational geometry, including (a) computing the width of a point set in 3-space, (b) computing the biggest stick in a simple polygon in the plane, and (c) computing the smallest-width annulus covering a planar point set. The solutions to these problems run in time $O(n^{17/11+\varepsilon})$, for any $\varepsilon > 0$, improving previous solutions that run in time $O(n^{8/5+\varepsilon})$. We also present data structures for (i) performing nearest-neighbor and related queries for fairly general collections of objects in 3-space and for collections of moving objects in the plane, and (ii) performing ray-shooting and related queries among n spheres or more general objects in 3-space. Both of these data structures require $O(n^{3+\varepsilon})$ storage and preprocessing time, for any $\varepsilon > 0$, and support polylogarithmic-time queries. These structures improve previous solutions to these problems.

1 Introduction

Let $\mathcal{F} = \{f_1, \ldots, f_n\}$ be a collection of n d-variate, possibly partially defined, functions, all algebraic of some constant maximum degree b (and if they are partially defined, their domains of definition are also described each by a constant number of polynomial equalities and inequalities of maximum degree b). Abusing the notation slightly, we will not distinguish between a function and its graph. The *lower envelope* $E_{\mathcal{F}}$ of $\mathcal{F}$ is defined as $E_{\mathcal{F}}(\mathbf{x}) = \min f_i(\mathbf{x})$, where the minimum is taken over all functions of $\mathcal{F}$ that are defined at $\mathbf{x}$. The *minimization diagram* $M_{\mathcal{F}}$ of $\mathcal{F}$ is the decomposition of $\mathbb{R}^d$ into maximal connected regions (of any dimension), called *cells*, so that within each cell the same subset of functions appears on the envelope $E_{\mathcal{F}}$. (A more detailed definition, treating also the case of partially-defined functions, is given in [28].) The *combinatorial complexity* of $M_{\mathcal{F}}$ and of $E_{\mathcal{F}}$ is the number of faces of all dimensions in $M_{\mathcal{F}}$.

Recently, there has been significant progress in the analysis of the combinatorial complexity of lower envelopes of multivariate functions [19, 28]. In particular, it was shown in [28] that the maximum com-

[*] Work on this paper by the first author has been supported by NSF Grant CCR-93-01259 and an NYI award. Work on this paper by the second author has been supported by NSF Grant CCR-92-11541. Work by the third author has been supported by NSF grant CCR-91-22103, by a Max-Planck Research Award, and by grants from the U.S.-Israeli Binational Science Foundation, the G.I.F. — the German-Israeli Foundation for Scientific Research and Development, and the Fund for Basic Research administered by the Israeli Academy of Sciences.

[†] Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129 USA

[‡] Department of Computer Science, Polytechnic University, Brooklyn, NY 11201 USA

[§] School of Mathematical Sciences, Tel Aviv University, Tel Aviv 69978, Israel, and Courant Institute of Mathematical Sciences, New York University, New York, NY 10012 USA

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

plexity of $M_{\mathcal{F}}$ is $O(n^{d+\varepsilon})$, for any $\varepsilon > 0$, where the constant of proportionality depends on ε, d and b. This result almost settles a major open problem and has already led to many applications [6, 19, 28]. However, less progress has been made on the corresponding algorithmic problem, which calls for the efficient construction of the lower envelope of such a collection $\mathcal{F}$, in time $O(n^{d+\varepsilon})$, for any $\varepsilon > 0$. The ideal output of an algorithm that computes the envelope is a data structure of size $O(n^{d+\varepsilon})$, which can return, for a given point $\mathbf{x} \in \mathbb{R}^d$, the identity of the function(s) attaining the envelope at $\mathbf{x}$ in logarithmic (or polylogarithmic) time. Weaker solutions might provide just an enumeration of the cells of $M_{\mathcal{F}}$, each cell augmented with its description as a semialgebraic set, and/or with a representative point lying in it, and possibly include explicit adjacency structure, representing all pairs of cells where one cell is contained in the closure of the other.

The paper [28] presents an algorithm for computing the lower envelope of bivariate functions, having the properties listed above, which runs in time $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$. Other algorithms with a similar performance are given in [9, 16]. The simplest solution to this problem, involving a deterministic divide-and-conquer algorithm, was recently presented in [3]. All these algorithms facilitate point location queries of the sort described above. Unfortunately, these methods fail in higher dimensions. For example, the technique of [28] relies on the existence of a *vertical decomposition* of the minimization diagram M_R of a sample R of r functions of $\mathcal{F}$, into a small number (that is, $O(r^{d+\varepsilon})$) of cells of *constant descriptional complexity.* Such decompositions exist (and are easy to compute) for $d = 2$, but their existence in higher dimensions is still an open problem.

In this paper we present an efficient algorithm for computing the vertices, edges, and two-dimensional faces of the lower envelopes of d-variate functions having the properties assumed above. We can use this algorithm to compute the entire lower envelope of a collection of tri-variate functions, which has all the desired characteristics; in particular, it supports queries for determining whether a given point lies above or below the envelope. The algorithm bypasses the problem of having to construct small-size vertical decomposition by applying the technique of Preparata and Tamassia [26] for point location in certain types of 3-dimensional subdivisions. This allows us to use a coarser decomposition of the minimization diagram, whose size is close to cubic.

Several recent papers [4, 11, 23] have studied a variety of geometric problems whose solution calls for the construction of, and searching in, lower or upper envelopes in 4-space. These applications fall into two main categories: *preprocess-for-queries* problems, which call for the construction of such an envelope, to be queried repeatedly later, and *batched* problems, where all the queries are known in advance, and the goal is to produce the answers to all of them efficiently. We will present some of these applications, as listed in the abstract, where our new algorithm for computing lower envelopes in four dimensions leads to improved solutions.

The paper is organized as follows. In Section 2 we present a general efficient randomized technique, which we believe to be of independent interest, for computing all the 0, 1, and 2-dimensional features of lower envelopes in any dimension. Then, in Section 3, we apply this algorithm to the case of tri-variate functions. This gives us an initial representation of the lower envelope of such a collection, which we then augment by additional features, resulting in a representation suitable for the application of the Preparata-Tamassia technique. In Sections 4 and 5 we present applications of our result to a variety of problems in computational geometry, as listed in the abstract.

2 Lower Envelopes in Arbitrary Dimension

In this section we present a randomized technique for computing a partial description of lower envelopes of d-variate functions, for $d \geq 2$, whose expected running time is $O(n^{d+\varepsilon})$, for any $\varepsilon > 0$. This technique only computes the vertices, edges, and 2-faces of the envelope, which is sufficient for the construction of envelopes in 4-space, as will be explained in Section 3 below.

Let $\mathcal{F} = \{f_1, \ldots, f_n\}$ be a collection of (partial) d-variate functions in $\mathbb{R}^{d+1}$, satisfying the conditions described in the introduction. We assume that the functions of $\mathcal{F}$ are in general position (see [28] for more details and for a discussion of this assumption). To compute the lower envelope $E_{\mathcal{F}}$ of $\mathcal{F}$, in the above sense, we proceed as follows. Let Σ be the family of all $(d-1)$-subsets of $\mathcal{F}$. We fix a subset $\sigma = \{f_1, \ldots, f_{d-1}\} \in \Sigma$, and let

$$\Pi^\sigma = \{\mathbf{x} \in \mathbb{R}^d \mid f_1(\mathbf{x}) = \cdots = f_{d-1}(\mathbf{x})\}.$$

Since the f_i's are assumed to be in general position, Π^σ is a two-dimensional surface (or a surface patch). For the sake of simplicity, we assume that Π^σ is connected and x_1x_2-monotone (i.e., any $(d-2)$-flat orthogonal to the x_1x_2-plane meets Π^σ in at most one point); otherwise we decompose it into a constant number of connected portions so that each portion is an x_1x_2-monotone surface patch, and work with each patch separately. For each $i \geq d$, let

$$\gamma_i = \{\mathbf{x} \in \Pi^\sigma \mid f_i(\mathbf{x}) = f_1(\mathbf{x}) = \cdots = f_{d-1}(\mathbf{x})\};$$

γ_i is a 1-dimensional curve, which partitions Π^σ into two (not necessarily connected) regions, K_i^+, K_i^-, where

$$\begin{aligned} K_i^+ &= \{\mathbf{x} \in \Pi^\sigma \mid f_i(\mathbf{x}) > f_1(\mathbf{x}) = \cdots = f_{d-1}(\mathbf{x})\} \\ K_i^- &= \{\mathbf{x} \in \Pi^\sigma \mid f_i(\mathbf{x}) < f_1(\mathbf{x}) = \cdots = f_{d-1}(\mathbf{x})\}. \end{aligned}$$

Then the intersection $Q^\sigma = \bigcap_{i \geq d} K_i^+$ is the portion of Π^σ over which the envelope $E_\mathcal{F}$ is attained by the functions of σ. The algorithm will compute the regions Q^σ, over all choices of $(d-1)$-tuples σ of functions, thereby yielding all the vertices, edges and 2-faces of $M_\mathcal{F}$ (because of the general position assumption, any such feature must show up as a feature of at least one of the regions Q^σ).

We compute Q^σ using a randomized incremental approach, similar to the ones described in [12, 16, 24, 25, 29]. Since the basic idea is by now fairly standard, we only give a brief overview of the algorithm, and refer the reader to [12, 29] for details. Let $\Gamma^\sigma = \{\gamma_i \mid d \leq i \leq n\}$. We first compute the set Γ^σ. (We need to assume an appropriate model of computation, in which any of the various primitive operations required by the algorithm can be performed in constant time. For example, we can assume the model used in computational real algebraic geometry [27], where each algebraic operation, involving a constant number of polynomials of constant maximum degree, can be performed exactly, using rational arithmetic, in constant time.) Next, we add the curves γ_i one by one in a random order, and maintain the intersection of the regions K_i^+ for the curves added so far. Let $(\gamma_d, \gamma_{d+1}, \ldots, \gamma_n)$ be the (random) insertion sequence, and let Q_i^σ denote the intersection of $K_d^+, \ldots, K_i^+$. We construct and maintain the 'vertical decomposition' $\tilde{Q}_i^\sigma$ of Q_i^σ, defined as the partitioning of each 2-face ϕ of Q_i^σ into 'pseudo-trapezoids', obtained by drawing, from each vertex of ϕ and from each locally x_1-extreme point on $\partial\phi$, a curve within ϕ orthogonal to the x_1-axis, and extend it (in both directions if necessary) until it hits $\partial\phi$ again. Each pseudo-trapezoid is defined by at most four curves of Γ^σ; conversely, any four or fewer curves of Γ^σ define a constant number of pseudo-trapezoidal cells, namely, those formed along Π^σ when only these curves are inserted. (Note that this construction is well-defined since Π^σ is an x_1x_2-monotone surface.) In the $(i+1)^{st}$-step we add K_{i+1}^+ and compute $\tilde{Q}_{i+1}^\sigma$ from $\tilde{Q}_i^\sigma$, using the technique described in [12].

The analysis of the expected running time of the algorithm proceeds along the same lines as described in [12, 29]. We define the *weight*, $w(\tau)$, of a pseudo-trapezoid τ, defined by the arcs of Γ^σ, to be the number of functions f_i, for $i = d, d+1, \ldots, n$ (excluding the up to 4 functions whose intersections with Π^σ *define* τ), such that $f_i(\mathbf{x}) \leq f_1(\mathbf{x}) = \cdots = f_{d-1}(\mathbf{x})$ for some point $\mathbf{x} \in \tau$.

As shown in [12], the cost of the above procedure is proportional to the number of pseudo-trapezoids that are created during the execution of the algorithm, plus the sum of their weights, plus an overhead term of $O(n^d)$, needed to prepare the collections of curves γ_i over all 2-dimensional intersection manifolds Π^σ. The analysis given below deals only with pseudo-trapezoids that are defined by exactly four such functions (plus the $d-1$ functions defining Π^σ), and easy and obvious modifications are necessary to handle all other pseudo-trapezoids.

Let T^σ denote the set of pseudo-trapezoids (or 'cells' for brevity) defined by four arcs of Γ^σ (again, see [12] for details), and let $T = \bigcup_{\sigma \in \Sigma} T^\sigma$. Each cell in T is defined by $d - 1 + 4 = d + 3$ functions of $\mathcal{F}$. In what follows we implicitly assume that the specification of a cell $\tau \in T$ includes the $d+3$ functions defining τ, where there is a clear distinction between the first $d-1$ functions (constituting the set σ), and the last four functions (defining the four curves that generate τ along Π^σ). For an integer $k \geq 0$ and a subset $R \subseteq \mathcal{F}$, let $T_k(R) \subseteq T$ (resp. $T_{\leq k}(R) \subseteq T$) denote the set of cells, defined by $d+3$ functions of R, as above, with weight k (resp. at most k). Let $N_k(R) = |T_k(R)|$ and $N_k(r) = \max_R N_k(R)$, where the maximum is taken over all subsets R of $\mathcal{F}$ of size r. Similarly, we define $N_{\leq k}(R)$ and $N_{\leq k}(r)$. Since each cell of $T_0(R)$ lies on the lower envelope of R, it follows that $N_0(r) = O(r^{d+\varepsilon})$. Adapting the analysis technique of Clarkson and Shor [15], we have

Lemma 2.1 *The probability that a cell $\tau \in T_k(\mathcal{F})$ is created during the incremental construction of Q^σ, where $\sigma \in \Sigma$ is the tuple for which Π^σ contains τ, is $1/\binom{k+4}{4}$.*

Lemma 2.2 *For any $0 \leq k \leq n$ and for any $\varepsilon > 0$,*

$$N_{\leq k}(n) = O((k+1)^{3-\varepsilon} n^{d+\varepsilon}).$$

For a cell $\tau \in T$, let A_τ be the event that $\tau \in \tilde{Q}_i^\sigma$, for the tuple $\sigma \in \Sigma$ for which Π^σ contains τ, and for some $d \leq i \leq n$. The expected running time of the algorithm, over all choices of $(d-1)$-tuples of functions, is thus proportional to

$$\sum_{\tau \in T} (w(\tau) + 1) \cdot \Pr[A_\tau] + O(n^d)$$

$$= \sum_{k\geq 0}\sum_{\tau\in T_k(\mathcal{F})}(k+1)\cdot \Pr[A_\tau] + O(n^d)$$

$$= \sum_{k=0}^{n-d-3}\frac{(k+1)N_k(\mathcal{F})}{\binom{k+4}{4}} + O(n^d)\ ,$$

where the last inequality follows from Lemma 2.1. Since $N_k(\mathcal{F}) = N_{\leq k}(\mathcal{F}) - N_{\leq(k-1)}(\mathcal{F})$, we obtain, using Lemma 2.2,

$$\sum_{k=0}^{n-d-3}\frac{(k+1)N_k(\mathcal{F})}{\binom{k+4}{4}}$$

$$= N_0(\mathcal{F}) + 24\sum_{k=1}^{n-d-3}\frac{N_{\leq k}(\mathcal{F}) - N_{\leq(k-1)}(\mathcal{F})}{(k+4)(k+3)(k+2)}$$

$$= O\left(n^{d+\varepsilon} + \sum_{k=1}^{n-d-4}\frac{k^{3-\varepsilon}n^{d+\varepsilon}}{(k+4)(k+3)(k+2)(k+1)}\right)$$

$$= O\left(n^{d+\varepsilon}\cdot \sum_{k=1}^{n-d-4}\frac{1}{k^{1+\varepsilon}}\right) = O(n^{d+\varepsilon})\ .$$

We thus obtain:

Theorem 2.3 *The vertices, edges, and 2-faces of the lower envelope of n (partial) d-variate functions, satisfying the conditions stated in the introduction, can be computed in randomized expected time $O(n^{d+\varepsilon})$, for any $\varepsilon > 0$.*

3 Envelopes in Four Dimensions

We next apply the results of the preceding section to obtain an efficient algorithm for constructing the lower envelope of a collection $\mathcal{F}$ of n tri-variate functions, satisfying the assumptions made above, in the following strong sense: One can preprocess $\mathcal{F}$, in randomized expected time $O(n^{3+\varepsilon})$, for any $\varepsilon > 0$, into a data structure of size $O(n^{3+\varepsilon})$ that supports queries of the form: given a point $w \in \mathbb{R}^3$, compute the function(s) attaining $E_\mathcal{F}$ at w; each query can be answered in $O(\log^2 n)$ time.

We first apply the algorithm summarized in Theorem 2.3, to compute the vertices, edges, and 2-faces of the minimization diagram $M_\mathcal{F}$ of $\mathcal{F}$. We next partition the cells of $M_\mathcal{F}$ into 'monotone' subcells, in the sense of Lemma 3.1 below, to obtain a refinement $M'_\mathcal{F}$ of $M_\mathcal{F}$. We then use the point-location technique of Preparata and Tamassia [26] to produce the data structure representing the lower envelope in the above manner.

We define and construct $M'_\mathcal{F}$ as follows. We mark, along each 2-face F of $M_\mathcal{F}$, the locus γ_F of all points of F which are either singular or have a vertical tangency (in the z-direction). The arcs γ_F, for all 2-faces F, lie along $O(n^2)$ curves in $\mathbb{R}^3$, each being the xyz-projection of the locus of all singular points or points with z-vertical tangency along some 2-manifold $f_i = f_j$, for a pair of indices $i \neq j$, or along the boundary of some f_i. We consider below only the former case; the latter case can be handled in almost the same (and, actually, simpler) manner. Let δ be one of these curves. We consider the 2-dimensional surface V_δ, within the xyz-space, obtained as the union of all lines passing through points of δ and parallel to the z-axis; let V_δ^+, V_δ^- denote the portions of V_δ that lie, respectively, above and below δ. (Note that V_δ may have self-intersections along some vertical lines, along which V_δ^+ and V_δ^- are not well-defined; we omit here details of the (rather easy) handling of such cases.)

Let δ_0 be the portion of δ over which the functions f_i and f_j attain the envelope $E_\mathcal{F}$. Clearly, δ_0 is the union of all arcs γ_F that are contained in δ, and each such edge of $E_\mathcal{F}$ is contained in some connected component of δ_0, so the number of connected components in δ_0, over all intersection curves, is $O(n^{3+\varepsilon})$. Let w be a point in δ_0. Then the cell c of $M_\mathcal{F}$ lying immediately above w in z-direction is such that within c the envelope $E_\mathcal{F}$ is attained by either f_i or f_j. Thus the upward-directed z-vertical ray emanating from w leaves c (if at all) at a point that lies on a 2-manifold of the form $f_i = f_k$ or $f_j = f_k$. For fixed i and j, there are only $O(n)$ possible 2-manifolds of this kind. We compute the lower envelope $E^{(\delta)}$ of these $O(n)$ 2-manifolds, restricted to V_δ^+. It is easily seen that the complexity of $E^{(\delta)}$ is $O(\lambda_q(n))$, for some constant q depending on the maximum degree and shape of these 2-manifolds; $\lambda_q(n)$ is the maximum length of Davenport-Schinzel sequences of order q, and is close to linear for any fixed q [3]. We next take the portions of the graph of $E^{(\delta)}$ that lie over δ_0, and "etch" them along the corresponding 2-faces of $E_\mathcal{F}$. We apply a fully symmetric procedure within V_δ^-, for all curves δ. The overall combinatorial complexity of all the added curves is thus $O(n^2\lambda_q(n) + n^{3+\varepsilon}) = O(n^{3+\varepsilon})$, and they can be computed in $O(n^{3+\varepsilon})$ time, as is easily verified.

Let $M'_\mathcal{F}$ denote the refined cell decomposition of $\mathbb{R}^3$, obtained by adding to $M_\mathcal{F}$ the arcs γ_F, the etched arcs of the upper and lower envelopes in the vertical manifolds V_δ, and the z-vertical walls (i.e., union of all z-vertical segments) contained in V_δ and connecting the arcs γ_F to the etched arcs. As just argued, the combinatorial complexity of $M'_\mathcal{F}$ is still $O(n^{3+\varepsilon})$, and $M'_\mathcal{F}$ has the following crucial property:

Lemma 3.1 *For each 3-cell c of $M'_\mathcal{F}$, every connected component of a cross section of c by a plane* *parallel to the xz-plane is x-monotone.*

Let $M'_{\mathcal{F}}(y_0)$ denote the cross section of $M'_{\mathcal{F}}$ by the plane $y = y_0$. Lemma 3.1 implies that $M'_{\mathcal{F}}(y_0)$ is an x-monotone planar subdivision, for each y_0. Hence, if we orient the edges of $M'_{\mathcal{F}}$ in the positive x-direction, add a pair of nominal points s, t at $x = -\infty$ and at $x = +\infty$, respectively, and connect them to the appropriate unbounded edges of $M'_{\mathcal{F}}(y_0)$, this map becomes a planar st-graph, in the notation of [26]. (Note that the vertical decomposition that generates $M'_{\mathcal{F}}$ from $M_{\mathcal{F}}$ may create z-vertical edges; if these edges are oriented consistently, say in the upward z-direction, then $M'_{\mathcal{F}}$ remains an st-graph.) We denote this st-graph also by $M'_{\mathcal{F}}(y_0)$. It is easy to verify the following statement.

Lemma 3.2 *Let I be an open interval of the y-axis that does not contain the y-coordinate of any vertex of $M'_{\mathcal{F}}$, or of any local y-extremum along any edge of $M'_{\mathcal{F}}$. Then, for each $y_1, y_2 \in I$, the st-graphs $M'_{\mathcal{F}}(y_1)$, $M'_{\mathcal{F}}(y_2)$ are isomorphic, as labeled embedded planar st-graphs.*

We are now in a position to apply the point location technique of Preparata and Tamassia [26]. They show that, given a 3-dimensional subdivision of combinatorial complexity N, it can be preprocessed in time $O(N \log^2 N)$ into a data structure of size $O(N \log^2 N)$, which supports $O(\log^2 N)$-time point-location queries in the given subdivision. However, for this to hold, the subdivision must have the properties that (a) (the 1-skeleton of) each cross section of the subdivision by a plane parallel to the xz-plane is a planar st-graph, whose faces are all monotone in the x-direction and whose edges are all oriented in the positive x-direction (or, for vertical edges, in the positive z-direction), and (b) at each point where the cross section, as an st-graph, changes, the graph can be updated by a constant number of operations from the following collection: (1) insertion of a vertex in the middle of an edge, or the complementary operation of deleting a vertex of in-degree and out-degree 1 and replacing the two incident edges by one, while maintaining x-monotonicity of the incident faces; (2) insertion of an edge partitioning an x-monotone face into two x-monotone subfaces, or, conversely, deletion of an edge and merging its two adjacent faces, provided their union remains x-monotone; and (3) merging two adjacent vertices into one vertex (collapsing the edge connecting them), or splitting a vertex into two vertices (forming a new edge between these vertices), again maintaining x-monotonicity.

It is easily verified that, indeed, each change occurring in the structure of the cross section $M_{\mathcal{F}}(\cdot)$, at any of the critical y values in Lemma 3.2, can be expressed as a constant number of operations of the types mentioned above.In summary, we thus obtain the following main result of the paper:

Theorem 3.3 *Let $\mathcal{F}$ be a given collection of n tri-variate, possibly partially defined, functions, all algebraic of constant maximum degree, and whose domains of definition (if they are partially defined) are each defined by a constant number of algebraic equalities and inequalities of constant maximum degree. Then, for any $\varepsilon > 0$, the lower envelope $E_{\mathcal{F}}$ of $\mathcal{F}$ can be computed in randomized expected time $O(n^{3+\varepsilon})$, and stored in a data structure of size $O(n^{3+\varepsilon})$, so that, given any query point $w \in \mathbb{R}^3$, we can compute $E_{\mathcal{F}}(w)$, as well as the function(s) attaining $E_{\mathcal{F}}$ at w, in $O(\log^2 n)$ time.*

4 Applications: Query Problems

In this section we apply Theorem 3.3 to two problems, involving preprocessing and querying certain collections of objects in 3-space.

4.1 Ray shooting amidst spheres

Given a collection $\mathcal{S}$ of n spheres in 3-space, we wish to preprocess $\mathcal{S}$ into a data structure that supports *ray-shooting* queries, each seeking the first sphere, if any, met by a query ray. This problem has recently been studied in [2, 5, 23]. The first two papers present a data structure that requires $O(n^{4+\varepsilon})$ storage and preprocessing, for any $\varepsilon > 0$, and answers a query in time $O(\log^2 n)$. The third paper [23] gives a rather elaborate and improved solution that requires $O(n^{3+\varepsilon})$ preprocessing and storage, for any $\varepsilon > 0$, and answers a query in $O(n^{\varepsilon})$ time. These algorithms use the technique described in [1], and reduce the problem to that of determining whether a query segment e intersects any sphere of $\mathcal{S}$. Then, using a multi-level data structure, they reduce the problem to that of detecting whether the line λ containing the segment e intersects any sphere of $\mathcal{S}$. As observed in [5, 23], this problem can be further reduced to point location in the upper envelope of a set of certain tri-variate functions, as follows. Let π be the plane passing through λ and orthogonal to the vertical plane V passing through λ. Let π^+ (resp. π^-) be the half-space lying above (resp. below) π. Let S be a sphere whose center lies in π^+ and which intersects V in a disc D. Then the center of D lies above λ, so either λ intersects S or passes below S, in the sense that λ and S are disjoint and there is a point on λ that lies vertically below a point in S. A similar property holds if the center of S lies in π^-. We preprocess the centers of spheres of $\mathcal{S}$ into a half-space range-searching data structure, of size $O(n^{3+\varepsilon})$, for a query λ, we can decompose $\mathcal{S}$ into $O(1)$ canonical subsets, so that, within each subset, either the centers of all spheres lie in π^+, or they all lie in π^-. Let us consider the case when the centers of all spheres lie in π^+.

Hence, we need to solve the following subproblem: Given a set $\mathcal{S}$ of n spheres in 3-space, and given a query line λ with the property that for each sphere $S \in \mathcal{S}$, either λ intersects S or there is no point of S lying vertically below λ, determine whether λ intersects any sphere of $\mathcal{S}$. We reduce this problem to point location in the lower envelope of certain trivariate functions, as follows. We can parametrize a line λ in 3-space by four parameters $(\xi_1, \xi_2, \xi_3, \xi_4)$, so that the equations defining λ are $y = x\xi_1 + \xi_2$, $z = x\xi_3 + \xi_4$. (We assume here that λ is not parallel to the yz-plane; such lines can be handled in a different, and much simpler manner.) For each sphere $S \in \mathcal{S}$, define a function $\xi_4 = F_S(\xi_1, \xi_2, \xi_3)$, so that the line $\lambda(\xi_1, \xi_2, \xi_3, \xi_4)$ is tangent to S from below (F_S is only partially defined, and we put $F_S = +\infty$ when it is undefined; it is easily checked that F_S is algebraic of bounded degree, and that its boundary is also algebraic of bounded degree). Let Φ be the lower envelope of the functions F_S, for $S \in \mathcal{S}$. Then a query line $\lambda(\xi_1, \xi_2, \xi_3, \xi_4)$ having the above properties misses all spheres of $\mathcal{S}$ if and only if $\xi_4 < \Phi(\xi_1, \xi_2, \xi_3)$. Thus, by Theorem 3.3, one can answer such queries in time $O(\log^2 n)$, using $O(n^{3+\varepsilon})$ preprocessing time and storage, for any $\varepsilon > 0$. Plugging the half-space range searching data structure and the point location data structure into the multi-level data structure described in [5, 23], we obtain a data structure for the segment-emptiness problem. A closer analysis of this data structure shows that the preprocessing time and storage of the overall data structure is still $O(n^{3+\varepsilon})$, for any $\varepsilon > 0$, and that the query time is $O(\log^2 n)$. Finally, plugging this data structure for the segment-emptiness query into the general ray-shooting technique by Agarwal and Matoušek [1], we obtain a final data structure, still requiring near-cubic storage and preprocessing, using which one can answer a ray-shooting query in time $O(\log^4 n)$. That is, we have shown:

Theorem 4.1 *A set $\mathcal{S}$ of n spheres in $\mathbb{R}^3$ can be preprocessed in randomized expected time $O(n^{3+\varepsilon})$ into a data structure of size $O(n^{3+\varepsilon})$, for any $\varepsilon > 0$, so that a ray shooting query can be answered in time $O(\log^4 n)$.*

4.2 Nearest neighbor queries

Let $\mathcal{S} = \{S_1, \ldots, S_n\}$ be a collection of n pairwise-disjoint convex objects ('sites') in 3-space, each having *constant description complexity*, namely, each defined by a constant number of algebraic equalities and inequalities of constant maximum degree. We wish to compute the *Voronoi diagram Vor*($\mathcal{S}$) of $\mathcal{S}$, and preprocess it for efficient point location queries. That is, each query specifies a point w in 3-space and seeks the site of $\mathcal{S}$ nearest to w (say, under the Euclidean distance). This is a generalization to 3-space of the classical *Post Office Problem.*

As observed in [18], the problem is equivalent to the computation of the following lower envelope in 4-space. For each $S \in \mathcal{S}$, define $F_S(x, y, z)$ to be the Euclidean distance from (x, y, z) to S, and let Φ be the lower envelope of these functions. Then, given a query point (x, y, z), the site(s) $S \in \mathcal{S}$ nearest to w are those for which F_S attains Φ at (x, y, z). Thus, by Theorem 3.3, this can be done using $O(n^{3+\varepsilon})$ preprocessing time and storage, for any $\varepsilon > 0$, and each query can be answered in $O(\log^2 n)$ time. (Note that Theorem 3.3 is indeed applicable here, because the functions F_S are all (piecewise) algebraic of constant maximum degree, as is easy to verify from the conditions assumed above.)

This is a fairly general framework, and admits numerous generalizations, e.g., we may replace the Euclidean distance by other distances, perform queries with objects other than points (as long as the location of a query object can be specified by only 3 real parameters; this is the case, e.g., if the query objects are translates of some rigid convex object), etc. An interesting generalization is to *dynamic* nearest-neighbor queries in the plane, where each object of $\mathcal{S}$ moves along some given trajectory, and each query (x, y, t) asks for the object of $\mathcal{S}$ nearest to the point (x, y) at time t. Using the same approach as above, this can be done, under appropriate assumptions on the shape and motion of the objects of $\mathcal{S}$, with $O(n^{3+\varepsilon})$ preprocessing time and storage, for any $\varepsilon > 0$, and $O(\log^2 n)$ query time.

Remark: There is a prevailing conjecture that the complexity of generalized Voronoi diagrams in 3-space, and of dynamic Voronoi diagrams in the plane, is only near-quadratic, under reasonable assumptions concerning the distance function and the shape of the sites (and of their motions). This was indeed proved recently in [14] for the case where the sites are lines in 3-space and the distance function is induced by a convex polyhedron. If this conjecture is established, then the above algorithm, of near-cubic cost, is far from being optimal, and will need to be improved considerably.

5 Applications: Batched Problems

We next consider *batched* applications. These applications solve a variety of rather unrelated problems, but they all involve an almost identical subproblem, where lower (or upper) envelopes in 4-space play a role. To avoid repetition, we describe in detail only one application, and then state the improved bounds for the other applications without proof, referring the reader to the relevant literature.

The following applications are based on the *parametric search* technique of Megiddo [22], which, in our case, requires a parallel algorithm for computing the lower envelope of a set of surfaces. However, the algorithm presented above is highly unparallelizable, because it uses an incremental insertion procedure of regions into 2-D arrangements, and later uses a plane-sweep in 3-space. To finesse this difficulty, we apply the parametric search technique with an additional twist that avoids the need to use a parallel algorithm.

5.1 Width in 3-space

Let S be a set of n points in 3-space. The width of S is the smallest distance between a pair of parallel planes so that the closed slab between the planes contains S. In two dimensions, the problem can be easily solved in $O(n \log n)$ time. An $O(n^2)$-time algorithm for three dimensions was presented in [21]. Recently, Chazelle et al. [11] presented an $O(n^{8/5+\varepsilon})$-time algorithm, for any $\varepsilon > 0$. In this subsection, we present an improved randomized algorithm, whose expected running time is $O(n^{17/11+\varepsilon}) = O(n^{1.546})$.

Clearly, it suffices to compute the width of the convex hull of S, so assume that the points of S are in convex position, and that the convex hull $\mathcal{P}$ of S is given. It is known that any two planes defining the width of S are such that either one touches a face of $\mathcal{P}$ and one touches a vertex of $\mathcal{P}$, or each of them touches an edge of $\mathcal{P}$; see [11, 21]. The first case can be handled in $O(n \log n)$ time [11, 21]. The difficult case is to compute the smallest distance between a pair of parallel planes supporting $\mathcal{P}$ at two 'antipodal' edges, because, in the worst case, there can be $\Theta(n^2)$ such pairs of edges. Chazelle et al. [11] presented an $O(n^{8/5+\varepsilon})$-time algorithm to find the smallest distance, using the parametric search technique. We will present a randomized algorithm, whose expected running time is $O(n^{17/11+\varepsilon})$, for any $\varepsilon > 0$, using a somewhat different approach.

Let $\mathcal{M}$ denote the *Gaussian diagram* (or *normal diagram*) of $\mathcal{P}$. $\mathcal{M}$ is a spherical map on the unit sphere $\mathbb{S}^2 \subset \mathbb{R}^3$. The vertices of $\mathcal{M}$ are points on $\mathbb{S}^2$, each being the outward normal of a face of $\mathcal{P}$, the edges of $\mathcal{M}$ are great circular arcs, each being the locus of the outward normal directions of all planes supporting $\mathcal{P}$ at some fixed edge, and the faces of $\mathcal{M}$ are regions, each being the locus of the outward normal directions of all planes supporting $\mathcal{P}$ at a vertex. $\mathcal{M}$ can be computed in linear time from $\mathcal{P}$. Let $\mathcal{M}'$ denote the spherical map $\mathcal{M}$ reflected through the origin, and consider the superposition of $\mathcal{M}$ and $\mathcal{M}'$. It suffices to consider the top parts of $\mathcal{M}$ and $\mathcal{M}'$, i.e., their portions within the hemisphere $z \geq 0$. Each intersection point between an edge of $\mathcal{M}$ and an edge of $\mathcal{M}'$ gives us a direction $\mathbf{u}$ for which there exist two parallel planes orthogonal to $\mathbf{u}$ and supporting $\mathcal{P}$ at two so-called antipodal edges. Thus the problem reduces to that of finding an intersection point for which the distance between the corresponding parallel supporting planes is minimized.

We centrally project the edges of (the top portions of) $\mathcal{M}$ and $\mathcal{M}'$ onto the plane $z = 1$. Each edge projects to a line segment or a ray. Let $\mathcal{E}$ and $\mathcal{E}'$ be the resulting set of segments and rays in the plane. Using the algorithm described in [10], we can decompose $\mathcal{E} \times \mathcal{E}'$, in time $O(n \log^2 n)$ into a family of 'canonical subsets'

$$\mathcal{F} = \{(\mathcal{E}_1, \mathcal{E}'_1), (\mathcal{E}_2, \mathcal{E}'_2), \ldots, (\mathcal{E}_t, \mathcal{E}'_t)\}, \qquad (1)$$

such that

(i) $\mathcal{E}_i \subseteq \mathcal{E}$ and $\mathcal{E}'_i \subseteq \mathcal{E}'$;

(ii) $\sum_{i=1}^{t}(|\mathcal{E}_i| + |\mathcal{E}'_i|) = O(n \log^2 n)$;

(iii) each segment in $\mathcal{E}_i$ intersects every segment of $\mathcal{E}'_i$; and

(iv) for every pair e, e' of intersecting segments in $\mathcal{E} \times \mathcal{E}'$, there is an i such that $e \in \mathcal{E}_i$ and $e' \in \mathcal{E}'_i$.

By property (iv), it suffices to consider each pair $(\mathcal{E}_i, \mathcal{E}'_i)$ separately. Let $\mathcal{L}_i$ (resp. $\mathcal{L}'_i$) denote the set of lines containing the edges of $\mathcal{P}$ corresponding to the edges of $\mathcal{E}_i$ (resp. $\mathcal{E}'_i$). By construction, all lines of $\mathcal{L}_i$ lie above all lines of $\mathcal{L}'_i$. Property (iii) implies that every pair of parallel planes π, π', where π contains a line λ of $\mathcal{L}_i$ and π' contains a line λ' of $\mathcal{L}'_i$, are supporting planes of $\mathcal{P}$ as well. Hence, we want to compute the smallest distance between two such planes. Since the distance between π and π', as above, is equal to the distance between λ and λ', it follows that this problem is equivalent to that of computing the closest pair of lines in $\mathcal{L}_i \times \mathcal{L}'_i$. Hence, we need to solve the following problem: Given a set $\mathcal{L}$ of m 'red' lines and another set $\mathcal{L}'$ of n 'blue' lines in $\mathbb{R}^3$, such that all red lines lie above all blue lines, compute the closest pair of lines in $\mathcal{L} \times \mathcal{L}'$. For any pair of lines $\ell, \ell' \in \mathbb{R}^3$, let $d(\ell, \ell')$ be the Euclidean distance between them, and let $d(\mathcal{L}, \mathcal{L}') = \min_{\ell \in \mathcal{L},\, \ell' \in \mathcal{L}'} d(\ell, \ell')$.

We first describe a simple randomized divide-and-conquer algorithm for computing $d(\mathcal{L}, \mathcal{L}')$, whose expected running time is $O(n^{3+\varepsilon} + m \log^2 n)$. If $m = O(1)$, we compute $d(\ell, \ell')$ for all pairs $\ell \in \mathcal{L}$, $\ell' \in \mathcal{L}'$, and choose the minimum distance. Otherwise, the algorithm performs the following steps:

(i) Choose a random subset $R_1 \subseteq \mathcal{L}$ of $m/2$ red lines; each subset of size $m/2$ is chosen with equal probability.

(ii) Solve the problem recursively for $(R_1, \mathcal{L}')$. Let $\delta_1 = d(R_1, \mathcal{L}')$.

(iii) Compute the set $R_2 = \{\ell \in \mathcal{L} \setminus R_1 \mid d(\ell, \mathcal{L}') < \delta_1\}$.

(iv) Compute $d(\ell, \ell')$ for all pairs $\ell \in R_2$, $\ell' \in \mathcal{L}'$, and output the minimum distance (or output δ_1 if R_2 is empty).

For a line $\ell' \in \mathcal{L}'$, let $R^{(\ell')} = \{\ell \in \mathcal{L} \mid d(\ell, \ell') < \delta_1\}$, so that $R_2 = \bigcup_{\ell' \in \mathcal{L}'} R^{(\ell')}$. Using a standard probabilistic argument, we can show that the expected size of $R^{(\ell')}$ is $O(1)$, for each $\ell' \in \mathcal{L}'$. Therefore the expected size of R_2 is $O(n)$. Consequently, the expected running time of Step (iv) is $O(n^2)$. The only non-trivial step in the above algorithm is Step (iii). We compute R_2 as follows. We map each line $\ell \in \mathcal{L}$ to a point $\psi(\ell) = (a_1, a_2, a_3, a_4)$ in $\mathbb{R}^4$, where $y = a_1 x + a_2$ is the equation of the xy-projection of ℓ, and $z = a_3 u + a_4$ is the equation of ℓ in the vertical plane $y = a_1 x + a_2$ (where u denotes the axis orthogonal to the z-axis). We can also map a line ℓ' to a surface $\gamma = \gamma(\ell')$ which is the locus of all points $\psi(\ell)$ such that $d(\ell, \ell') = \delta_1$ and ℓ lies above ℓ'. Removal of γ partitions $\mathbb{R}^4$ into two subsets, one, denoted by γ^-, consisting of points for which the corresponding lines either lie below ℓ', or lie above ℓ' and $d(\ell, \ell') < \delta_1$, and the other subset, γ^+, consisting of points whose corresponding lines lie above ℓ' and $d(\ell, \ell') > \delta_1$. Our choice of parameters ensures that $\gamma(\ell')$ is $x_1x_2x_3$-monotone, i.e., any line parallel to the x_4-axis intersects $\gamma(\ell')$ in at most one point. Hence, $\gamma(\ell')$ is the graph of a function $x_4 = f_{\ell'}(x_1, x_2, x_3)$. Let

$$F(x_1, x_2, x_3) = \max_{\ell' \in \mathcal{L}'} f_{\ell'}(x_1, x_2, x_3)$$

be the upper envelope of the set $\{f_{\ell'} \mid \ell' \in \mathcal{L}'\}$. For a line $\ell \in \mathcal{L}$ with $\psi(\ell) = (a_1, a_2, a_3, a_4)$, we have $a_4 \geq F(a_1, a_2, a_3)$ if and only if $d(\{\ell\}, \mathcal{L}') \geq \delta_1$. The problem of computing R_2 thus reduces to computing the set of points $\psi(\ell)$, for $\ell \in \mathcal{L} \setminus R_1$, that lie below the upper envelope F. By Theorem 3.3, this can be accomplished in time $O(n^{3+\varepsilon} + m \log^2 n)$, for any $\varepsilon > 0$. The total time spent in Steps (iii) and (iv) is thus $O(n^{3+\varepsilon} + m \log^2 n)$. Let $T(m, n)$ denote the maximum expected running time of the algorithm. Then

$$T(m, n) \leq T(\frac{m}{2}, n) + O(n^{3+\varepsilon} + m \log^2 n)\,.$$

The solution to this recurrence is easily seen to be $O(n^{3+\varepsilon} \log m + m \log^2 n) = O(n^{3+\varepsilon'} + m \log^2 n)$, for any $\varepsilon' > \varepsilon$. Hence, we can conclude:

Lemma 5.1 *Given a set $\mathcal{L}$ of m lines and another set $\mathcal{L}'$ of n lines in $\mathbb{R}^3$, such that all lines in $\mathcal{L}$ lie above all lines of $\mathcal{L}'$, the closest pair of lines in $\mathcal{L} \times \mathcal{L}'$ can be computed in time $O(n^{3+\varepsilon} + m \log^2 n)$, for any $\varepsilon > 0$.*

Next, we describe another algorithm for computing $d(\mathcal{L}, \mathcal{L}')$, which uses the above procedure as a subroutine. We first describe an algorithm for the 'fixed-size' problem that, given a parameter δ, determines whether $d(\mathcal{L}, \mathcal{L}') < \delta$, $d(\mathcal{L}, \mathcal{L}') = \delta$, or $d(\mathcal{L}, \mathcal{L}') > \delta$. Next, we apply the parametric search technique to this algorithm, with an additional twist, to compute $d(\mathcal{L}, \mathcal{L}')$.

Consider the fixed-size problem. If $m > n^{3+\varepsilon}$, the problem can be solved, in $O(m \log^2 n)$ time, by computing $\delta^* = d(\mathcal{L}, \mathcal{L}')$, using Lemma 5.1, and by comparing δ^* with δ. So assume that $m \leq n^3$.

We now map each line $\ell' \in \mathcal{L}'$ to the point $\psi(\ell')$, as defined above, and each line $\ell \in \mathcal{L}$ to a surface $\phi(\ell)$ which is the locus of all points $\psi(\ell')$ such that $d(\ell, \ell') = \delta$ and ℓ' lies below ℓ. (This is a dual representation of the problem, where the roles of $\mathcal{L}$ and of $\mathcal{L}'$ are interchanged.) The open region ϕ^- lying below $\phi(\ell)$ consists of points corresponding to lines ℓ' that lie below ℓ and satisfy $d(\ell, \ell') > \delta$. Again, each surface $\phi(\ell)$ is $x_1x_2x_3$-monotone, i.e., $\phi(\ell)$ is the graph of a function $x_4 = g_\ell(x_1, x_2, x_3)$. Let

$$G(x_1, x_2, x_3) = \min_{\ell \in \mathcal{L}} g_\ell(x_1, x_2, x_3)$$

be the lower envelope of these functions. By the same argument as above, $d(\mathcal{L}, \mathcal{L}') < \delta$ if and only if there is a point $\psi(\ell')$, for some $\ell' \in \mathcal{L}'$, that lies above the lower envelope G, and $d(\mathcal{L}, \mathcal{L}') = \delta$ if and only if no point corresponding to the lines of $\mathcal{L}'$ lies above G and at least one such point lies on G. To detect these two conditions, we proceed as follows.

(i) Fix a sufficiently large constant r. Choose a random subset $R \subset \mathcal{L}$ of size $t = cr \log r$, where c is some appropriate constant independent of r.

(ii) Let $\Gamma_R = \{g_\ell \mid \ell \in R\}$ be the set of t tri-variate functions corresponding to the lines of R, and let G_R be the lower envelope of Γ_R. Decompose the region of $\mathbb{R}^4$ lying below G_R into a collection $\Xi = \{\tau_1, \ldots, \tau_k\}$ of $k = O(t^4\beta(t)) = O(r^4\beta(r)\log^4 r)$ semialgebraic cells of constant description complexity each; here $\beta(r)$ is a slowly growing function whose exact form is determined by the algebraic degree of the surfaces in Γ. Such a decomposition can be obtained using the algorithms described in [4, 11]. It is based on a decomposition of $\mathbb{R}^3$ into a family Ξ' of $O(r^4\beta(r)\log^4 r)$ cells of constant description complexity, so that, within each cell, the same set of function graphs appear on the lower envelope G_R. The decomposition Ξ is then defined as the following collection of semi-unbounded 'prisms':

$$\Xi = \left\{\{(\mathbf{x}, z) \mid \mathbf{x} \in \Delta, z \le G_R(\mathbf{x})\} \mid \Delta \in \Xi'\right\}.$$

(iii) For each prism $\tau \in \Xi$, let $\mathcal{L}_\tau = \{\ell \in \mathcal{L} \mid \phi(\ell) \cap \tau \neq \emptyset\}$, and $\mathcal{L}'_\tau = \{\ell' \in \mathcal{L}' \mid \psi(\ell') \in \tau\}$.

(iv) If $\bigcup_{\tau\in\Xi} \mathcal{L}'_\tau \neq \mathcal{L}'$ (i.e., there is a line $\ell' \in \mathcal{L}'$ such that the point $\psi(\ell')$ lies above G_R), then return $\delta > \delta^*$. If there is a $\tau \in \Xi$ such that $|\mathcal{L}_\tau| > m/r + 1$, then go back to Step (i).

(v) For each prism $\tau \in \Xi$, if both $\mathcal{L}_\tau, \mathcal{L}'_\tau$ are nonempty, then solve the problem recursively for $\mathcal{L}_\tau$ and $\mathcal{L}'_\tau$. If there is a subproblem for which $\delta > d(\mathcal{L}_\tau, \mathcal{L}'_\tau)$, then return $\delta > \delta^*$. If there is no such subproblem, but there is one subproblem for which $\delta = \delta^*$, then return $\delta = \delta^*$. Finally, if none of these two cases occur, then return $\delta < \delta^*$.

The correctness of the algorithm follows from the above observations. As for the running time, the well-known results on random sampling [15, 20] imply that if c is chosen sufficiently large then, with high probability, $|\mathcal{L}_\tau| \le m/r + 1$ for every $\tau \in \Xi$. Hence, the expected number of times we have to go back to Step (i) from Step (iv) is only a constant. Let $T(m, n)$ denote the expected running time of the algorithm. Then

$$T(m,n) \le \sum_{i=1}^{k} T\Big(\frac{m}{r} + 1, n_i\Big) + O(m+n),$$

for $m < n^{3+\varepsilon}$, where $\sum_i n_i = n$. The solution of the above recurrence is easily seen to be $O(m^{8/11+\varepsilon'} n^{9/11+\varepsilon'} + m^{1+\varepsilon'} + n^{1+\varepsilon'})$, for any $\varepsilon' > \varepsilon$.

Next, we describe how to apply the parametric search technique to this algorithm. As mentioned earlier, we cannot use this algorithm directly in the parametric search paradigm, because we do not know how to parallelize the first algorithm. We therefore simulate the above sequential algorithm at $\delta = \delta^*$, with the additional twist that, instead of just simulating the solution of the fixed-size problem at δ^*, the algorithm will attempt to compute δ^* explicitly. Since r is chosen to be constant, the set Ξ can be computed by making only $r^{O(1)}$ implicit "sign tests" involving δ^*, thus only a constant number, $r^{O(1)}$, of solutions of the fixed-size problem are needed; see [4, 11] for details. It can be checked that the problem of determining whether a surface $\phi(\ell)$ intersects a prism $\tau \in \Xi$ or whether a point $\psi(\ell')$ lies in τ can be reduced to computing the signs of a constant number of univariate polynomials, each of constant degree, at δ^*. Let $\Pi = \{p_1(\delta), \ldots, p_s(\delta)\}$, $s = O(m+n)$, be the set of these polynomials, over all lines of $\mathcal{L}, \mathcal{L}'$ and over all prisms of Ξ. Let $\delta_1 < \cdots < \delta_u$ be the real roots of these polynomials. By a binary search over these roots we compute the largest root δ_α such that $\delta_\alpha \le d(\mathcal{L}, \mathcal{L}')$. Each step of the binary search involves comparing δ^* with some δ_i, which in turn involves solving an instance of the fixed size problem for some δ_i. If $\delta_\alpha = \delta^*$, we have found the value of δ^* so we stop right away. Thus, assume that $\delta_\alpha < \delta^*$. We can now easily resolve the signs of all polynomials of Π by evaluating them at $(\delta_\alpha + \delta_{\alpha+1})/2$. Once we have computed $\mathcal{L}_\tau, \mathcal{L}'_\tau$ for all $\tau \in \Xi$, we compute $d(\mathcal{L}_\tau, \mathcal{L}'_\tau)$ recursively and return $\min_{\tau\in\Xi} d(\mathcal{L}_\tau, \mathcal{L}'_\tau)$ as $d(\mathcal{L}, \mathcal{L}')$. (Note that $\bigcup_{\tau\in\Xi} \mathcal{L}'_\tau = \mathcal{L}'$, since we are simulating the algorithm at δ^*.) The correctness of the algorithm is established by the following lemma.

Lemma 5.2 *If none of the calls to the fixed size problem in Steps (i)–(iv) returns the value of $d(\mathcal{L}, \mathcal{L}')$, then $d(\mathcal{L}, \mathcal{L}') = \min_{\tau\in\Xi} d(\mathcal{L}_\tau, \mathcal{L}'_\tau)$.*

Proof: Let $d(\mathcal{L}, \mathcal{L}') = \delta^*$, and let $\ell \in \mathcal{L}$, $\ell' \in \mathcal{L}'$ be a pair of lines such that $d(\ell, \ell') = \delta^*$. Since we are simulating the fixed-size problem at $\delta = \delta^*$, no point corresponding to the lines of $\mathcal{L}'$ lies above the lower envelope of the surfaces defined by $\mathcal{L}$ (for $\delta = \delta^*$), i.e., there exists a prism $\tau \in \Xi$ such that $\ell' \in \mathcal{L}'_\tau$ (i.e., $\psi(\ell')$ lies in τ). If ℓ does not belong to $\mathcal{L}_\tau$, then $\psi(\ell')$ lies below $\phi(\ell)$, which implies that $d(\ell, \ell') > \delta^*$, a contradiction. Hence the pair (ℓ, ℓ') appears in the subproblem involving $\mathcal{L}_\tau$ and $\mathcal{L}'_\tau$. □

Let $T'(m, n)$ denote the maximum expected running time of the algorithm. Then we obtain the following recurrence

$$T'(m,n) \le \sum_{i=1}^{k} T'\Big(\frac{m}{r} + 1, n_i\Big) + r^{O(1)} \cdot T(m,n),$$

for $m < n^{3+\varepsilon}$. The solution of the above recurrence is also $O(m^{8/11+\varepsilon'} n^{9/11+\varepsilon'} + m^{1+\varepsilon'} + n^{1+\varepsilon'})$, for a different but still arbitrarily small constant $\varepsilon' > \varepsilon$. Hence, we can conclude

Lemma 5.3 *Given a set $\mathcal{L}$ of m lines and another set $\mathcal{L}'$ of n lines such that all lines in $\mathcal{L}$ lie above all lines of $\mathcal{L}'$, the closest pair of lines in $\mathcal{L} \times \mathcal{L}'$ can be computed in time $O(m^{8/11+\varepsilon}n^{9/11+\varepsilon} + m^{1+\varepsilon} + n^{1+\varepsilon})$, for any $\varepsilon > 0$.*

Finally, we apply Lemma 5.3 to all subsets $(\mathcal{E}_i, \mathcal{E}_i')$ of $\mathcal{F}$ (cf. (1)), and output the minimum of the distances obtained for each subproblem. This is equal to the minimum distance between any pair of parallel planes, each supporting an edge of $\mathcal{P}$. The total expected running time is $\sum_{i=1}^{t} O\left(|\mathcal{E}_i|^{8/11+\varepsilon}|\mathcal{E}_i'|^{9/11+\varepsilon} + |\mathcal{E}_i|^{1+\varepsilon} + |\mathcal{E}_i'|^{1+\varepsilon}\right) = O(n^{17/11+\varepsilon'})$, where ε' is yet another but still arbitrarily small positive constant. Putting everything together, we can conclude:

Theorem 5.4 *The width of a set of n points in $\mathbb{R}^3$ can be computed in randomized expected time $O(n^{17/11+\varepsilon})$, for any $\varepsilon > 0$.*

5.2 Biggest stick in a simple polygon

Let P be a simple polygon in the plane having n edges. We wish to find the longest line segment e that is contained in (the closed set) P. Chazelle and Sharir presented in [13] an $O(n^{1.99})$-time algorithm for this problem, which was later improved by Agarwal et al. [4] to $O(n^{8/5+\varepsilon})$, for any $\varepsilon > 0$. The latter paper reduces this problem, just as in the computation of the width of a point set in 3-space, to that of finding the extreme value of a certain function of two parameters, optimized over all intersection points between the edges of two planar maps. This problem, in turn, reduces to the problem of optimizing the function over all intersection points of pairs of lines in $\mathcal{L}_1 \times \mathcal{L}_2$, where $\mathcal{L}_1, \mathcal{L}_2$ are two appropriate families of lines. By regarding the lines of $\mathcal{L}_1$ as data points, and each line of $\mathcal{L}_2$ as inducing a certain function over the lines of $\mathcal{L}_1$, and by using an appropriate parametrization of these points and functions, the problem can be reduced to that of testing whether any point in a certain set of points in 4-space lies above the lower envelope of a certain collection of tri-variate functions. Combining the approach of [4] with the one described in the previous subsection, we can compute the longest segment e in expected time $O(n^{17/11+\varepsilon})$, for any $\varepsilon > 0$. Omitting all the details, which can be found in [4], we conclude:

Theorem 5.5 *One can compute the biggest stick that fits inside a simple polygon with n edges, in randomized expected time $O(n^{17/11+\varepsilon})$, for any $\varepsilon > 0$.*

5.3 Minimum-width annulus

Let S be a set of n points in the plane. We wish to find the (closed) annulus of smallest width that contains S, i.e., we want to compute two concentric disks D_1 and D_2 of radii r_1 and r_2, such that all points of S lie in the (closure of) $D_2 - D_1$ and $r_2 - r_1$ is minimized. This problem has been studied in [4]. It is known [4, 17] that the center of such an annulus is either a vertex of the closest-point Voronoi diagram, $\mathrm{Vor}_c(S)$, of S, a vertex of the farthest-point Voronoi diagram, $\mathrm{Vor}_f(S)$, of S, or the intersection point of an edge of $\mathrm{Vor}_c(S)$ and an edge of $\mathrm{Vor}_f(S)$. The difficult part is testing the intersection points of edges of the two diagrams, because there can be $\Theta(n^2)$ such points in the worst case. Following the same idea as in [4], which reduces the problem to that of batched searching of points relative to an envelope of functions in 4 dimensions, but using the technique described above for computing the width in 3-space, we obtain:

Theorem 5.6 *The smallest-width annulus containing a set of n points in the plane can be computed in randomized expected time $O(n^{17/11+\varepsilon})$, for any $\varepsilon > 0$.*

References

[1] P. Agarwal and J. Matoušek, Ray shooting and parametric search, *SIAM J. Comput.* 22 (1993), 794–806.

[2] P. Agarwal and J. Matoušek, Range searching with semialgebraic sets, *Proc. 17th Symp. on Mathematical Foundations of Computer Science*, 1992, 1–13.

[3] P. Agarwal, O. Schwarzkopf and M. Sharir, The overlay of lower envelopes and its applications, manuscript, 1993.

[4] P. Agarwal, M. Sharir and S. Toledo, New applications of parametric searching in computational geometry, to appear in *J. Algorithms.*

[5] P. Agarwal, L. Guibas, M. Pellegrini, and M. Sharir, Ray shooting among spheres, manuscript, 1992.

[6] P. Agarwal and M. Sharir, On the number of views of polyhedral terrains, *Proc. 5th Canadian Conf. Computational Geometry*, 1993, 55–61.

[7] B. Aronov and M. Sharir, Triangles in space, or: Building (and analyzing) castles in the air, *Combinatorica* 10 (1990), 137–173.

[8] J.D. Boissonnat, O. Devillers, R. Schott, M. Teillaud and M. Yvinec, Applications of random sampling to on-line algorithms in computational geometry, *Discrete Comput. Geom.* 8 (1992), 51–71.

[9] J.D. Boissonnat and K. Dobrindt, On-line randomized construction of the upper envelope of triangles and surface patches in $\mathbf{R}^3$, manuscript, 1993.

[10] B. Chazelle, H. Edelsbrunner, L. Guibas and M. Sharir, Algorithms for bichromatic line segment problems and polyhedral terrains, *Algorithmica* 11 (1994), 116–132.

[11] B. Chazelle, H. Edelsbrunner, L. Guibas and M. Sharir, Diameter, width, closest line pair, and parametric searching, *Discrete Comput. Geom.* 10 (1993), 183–196.

[12] B. Chazelle, H. Edelsbrunner, L. Guibas, M. Sharir and J. Snoeyink, Computing a single face in an arrangement of line segments and related problems, *SIAM J. Computing* 22 (1993), 1286–1302.

[13] B. Chazelle and M. Sharir, An algorithm for generalized point location and its applications, *J. Symbolic Computation* 10 (1990), 281–309.

[14] L.P. Chew, K. Kedem, M. Sharir and E. Welzl, Voronoi diagrams of lines in 3-space under a polyhedral convex distance function, in preparation.

[15] K. Clarkson and P. Shor, Applications of random sampling in computational geometry II, *Discrete Comput. Geom.* 4 (1989), 387–421.

[16] M. de Berg, K. Dobrindt and O. Schwarzkopf, On lazy randomized incremental construction, *Proc. 26th ACM Symp. on Theory of Computing*, 1994, to appear.

[17] H. Ebara, N. Fukuyama, H. Nakano and Y. Nakanishi, Roundness algorithms using the Voronoi diagrams, *First Canadian Conf. Computational Geometry*, 1989.

[18] H. Edelsbrunner and R. Seidel, Voronoi diagrams and arrangements, *Discrete Comput. Geom.* 1 (1986), 25–44.

[19] D. Halperin and M. Sharir, New bounds for lower envelopes in 3 dimensions, with applications to visibility in terrains, *Proc. 9th ACM Symp. on Computational Geometry*, 1993, 11–18. (Also to appear in *Discrete Comput. Geom.*)

[20] D. Haussler and E. Welzl, ϵ-nets and simplex range queries, *Discrete Comput. Geom.* 2 (1987), 127–151.

[21] M. Houle and G. Toussaint, Computing the width of a set, *Proc. 1st ACM Symp. Computational Geometry*, 1985, pp. 1–7.

[22] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM* 30 (1983), 852–865.

[23] S. Mohaban and M. Sharir, Ray shooting amidst spheres in three dimensions, manuscript, 1993.

[24] K. Mulmuley, A fast planar partition algorithm, I, *J. Symbolic Computing* 10 (1990), 253–280.

[25] K. Mulmuley, A fast planar partition algorithm, II, *J. ACM* 38 (1991), 74–103.

[26] F.P. Preparata and R. Tamassia, Efficient point location in a convex spatial cell-complex, *SIAM J. Comput.* 21 (1992), 267–280.

[27] J. Heintz, T. Recio and M.-F. Roy, Algorithms in real algebraic geometry and applications to computational geometry, in *Discrete and Computational Geometry: Papers from the DIMACS Special Year* (J.E. Goodman, R. Pollack and W. Steiger, Eds.), AMS Press, Providence, RI 1991, pp. 137–163.

[28] M. Sharir, Almost tight upper bounds for lower envelopes in higher dimensions, *Proc. 34th IEEE Symp. on Foundations of Computer Science,* 1993, 498–507. (Also to appear in *Discrete Comput. Geom.*)

[29] M. Sharir and P. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, to appear.

Query-Sensitive Ray Shooting

Joseph S. B. Mitchell[*] David M. Mount[†] Subhash Suri[‡]

1 Introduction

Ray shooting is the problem of determining the first intersection of a ray with a set of *obstacles.* We assume that obstacles are modeled as polygons in the plane, or generally polyhedra in higher dimensions. Since many ray shooting queries may be applied to a given set of obstacles, it is desirable to preprocess the obstacles so that query processing is as efficient as possible. The problem of ray shooting in a simple polygon with n vertices has been studied by [7, 6, 10]. In all cases, with linear space, ray shooting queries can be answered in $O(\log n)$ time. Ray shooting amidst a set of polygonal obstacles has been considered by [1, 6, 8, 10, 15]. In all these cases, space is within a polylogarithmic factor of linear, and the best query time is $O(\sqrt{n}\log n)$ query time.

One of the shortcomings of these approaches is that they fail to account for the geometric structure of the obstacle set and query ray. Analyses based solely upon input size cannot account for the fact that, for a fixed input size, there exist configurations of obstacles and rays for which solving ray shooting queries is intuitively very easy, and others for which it is quite a bit more complicated. Practical experience suggests that worst-case scenarios are rare for many applications. By concentrating on worst-case asymptotic performance, algorithm designers may overlook simple approaches that perform very well in the vast majority of practical situations, even though their worst-case performance can be quite poor. One approach for dealing with this phenomenon is to design algorithms for special-purpose classes of inputs, e.g. for convex, star-shaped, or monotone polygons. However, these approaches may not be useful if inputs do not satisfy these conditions. An alternative approach is to design algorithms with good expected-case performance. However, there does not seem to be a useful notion of "random" polygon or "random" polyhedron that captures the structure existing in many geometric applications.

In this paper we suggest a different approach. We present a fully general algorithm for the ray shooting problem, whose performance is not described as a function of input, but rather as a function of the complexity of a query with respect to an intrinsic parameter of the input. This parameter is called the *simple cover complexity*, and will be defined shortly. We feel that this parameter intuitively captures the geometric complexity of a given query. As in [6, 10], we construct a carefully designed subdivision of free space that has low "stabbing number" with respect to that portion of a query segment that lies within free space. This allows query processing to proceed by a simple "walk" through the subdivision.

We focus on a generalization of ray shooting, called *directed segment shooting* (or just *segment shooting*, for short). Given a set of obstacles and a directed line segment $\overline{pq}$, determine the first intersection (if any) with an obstacle for a point traveling along this segment from p

[*] jsbm@ams.sunysb.edu, Applied Math, SUNY, Stony Brook, NY 11794-3600. Partially supported by grants from Boeing Computer Services, Air Force Office of Scientific Research contract AFOSR-91-0328, and by NSF Grants ECSE-8857642 and CCR-9204585.

[†] mount@cs.umd.edu, Dept. of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Partially supported by NSF Grant CCR-9310705.

[‡] suri@bellcore.com, Bellcore, Room 2Q-358, 445 South Street, Morristown, NJ 07960.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

to q. If there is no intersection, the point q is returned. The reason for this formulation is that there are applications of ray shooting arising in motion simulation where a number of short segment queries are used to approximate more complex curved motion. Furthermore, in some interference-detection algorithms for virtual reality applications, one tracks the motion of an object through free-space by performing segment-shooting queries for each boundary edge of the moving object [12].

Before defining the complexity parameter referred to above, we give some definitions. Let P denote a finite collection of pairwise-disjoint polygonal obstacles in the plane, or, more generally, polyhedral obstacles in d-space. Because our algorithms are based on repeated subdivision of space into sufficiently simple regions, we need to make the *bounded incidence assumption* that each point is incident to at most a constant number, β, of faces of dimensions $\leq d-1$. Here *face* refers to boundary flats of all dimensions $\leq d-1$. Define a *ball* of radius r to be the (open) set of points that are less than distance r from a given center point. In the plane, a ball is *simple* if it intersects at most 2 edges of P. In dimension $d \geq 3$, the value 2 can be replaced with any constant $\geq \beta$. Given any $\epsilon > 0$, we say that a ball of radius r is *ϵ-strongly simple* if the ball with the same center and radius $(1+\epsilon)r$ is simple. Given ϵ, a *strongly simple cover* of a domain D, is a collection of ϵ-strongly simple balls whose union contains D. (Note that the balls themselves cover D, but their expansion by $(1+\epsilon)$ is simple.) Finally, the *simple cover complexity* of D (*with respect to* ϵ), $scc(D)$, is defined to be the cardinality of the smallest strongly simple cover of D. We will use the term *complexity* for short.

For example, in Fig 1, we show a strongly simple cover of size 4 for a line segment (shown as a dashed line). The ϵ value is 1/4. Solid circles are the balls of the cover, and dotted circles are balls of radius $(1+\epsilon)r$.

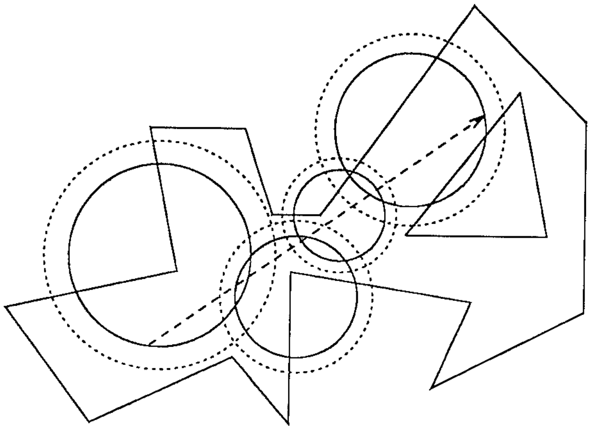


Figure 1: Strongly simple cover.

Intuitively, it is easy to see that a ray that is short and geometrically well separated from all obstacles will have a smaller complexity value than a ray that grazes the boundary of the obstacles. In this sense, we feel that this complexity measure is a reasonable reflection of the intrinsic difficulty of solving a ray shooting query. Observe that complexity is invariant under orthogonal transformations of the plane; however, it is not invariant under general affine transformations, since balls are not preserved.

We present two results. For ray shooting queries in arbitrary dimensions, we analyze the performance of a simple variant of an existing data structure, called a *PM k-d tree* [16], with respect to simple cover complexity. Samet [16], gives an analysis of the maximum depth of this tree in terms of the minimum separation distance between nonadjacent vertices and edges. We show that the size of the data structure and the time complexity of segment shooting queries can be related to the simple cover complexity of the obstacle set and the query segment, respectively. In the plane, we strengthen these results. Given a set of polygonal obstacles with a total of n vertices, we show how to to build a subdivision of size $O(n)$ in $O(n \log n)$ time, such that segment shooting queries can be answered in time $O(\log n + scc(s))$, where $scc(s)$ is the simple cover complexity of the obstacle-free portion of the query segment.

Because we terminate our search along a query ray as soon as it hits an obstacle, it is possible to make a further improvement in the planar analysis. We say that two points can *reach* one another if there is a path between them that does not intersect any obstacle. A region of space is *connected* if all points within this region are mutually reachable, and a *connected component* of the region is a maximally connected subset of the region. We can modify the definition of simplicity, by saying that a ball that contains some point p is *C-simple* (where "C" stands for connected) if p's connected component in the ball is bounded by at most two obstacle edges. We can modify the definition of complexity using C-simplicity, to derive a notion of *C-complexity*. It is easy to see that any ball that is simple is also C-simple, and thus the C-complexity of an object is not greater than its complexity. Consider, for example, the line segment shown as a dashed line in Fig. 2. Its standard complexity would be quite high, but because the connected components of the two covering balls containing this segment are simple, its C-complexity is only 2. Our results in the plane hold under this connected notion of simple cover complexity.

Before describing our algorithms we mention one technical fact. The definition of simplicity is based on a parameter $\epsilon > 0$. In the following theorem we observe that the choice of ϵ affects the complexity of a domain by only a constant factor, and so henceforth we exclude references to ϵ when deriving asymptotic bounds. Its proof is straightforward, and is omitted from this version.

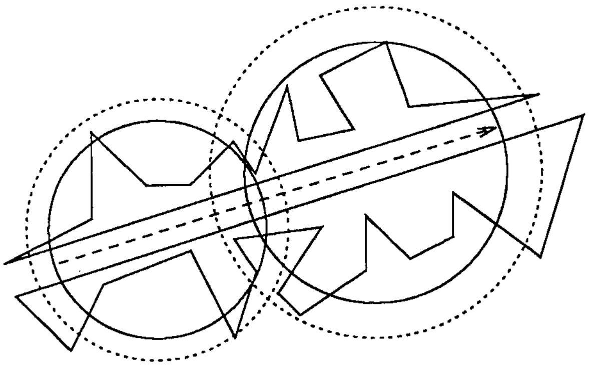


Figure 2: C-complexity.

Lemma 1.1 *Let $0 < \epsilon < \epsilon'$, and let D be some domain. Let $scc(D)$ denote the simple cover complexity of D with respect to ϵ, and let $scc'(D)$ denote the simple cover complexity with respect to ϵ'. Then there is a constant c (depending on ϵ, ϵ' and dimension) such that*

$$scc(D) \le scc'(D) \le c \cdot scc(D).$$

2 A Simple Approach

In this section we give a query-sensitive analysis of a simple, practical decomposition strategy, which illustrates the essential features of our method. The decomposition strategy is similar to one described by Samet [16], called a *PM k-d tree.*

We begin by assuming that we are given a set of polyhedral obstacles in d-dimensional space. As mentioned in the introduction, we assume that faces bounding these obstacles are simple in the sense that each point is incident to at most a constant number, β, of faces of dimension $\le d-1$. We also assume that the obstacles are contained within a minimal bounding hypercube. (Such a hypercube is easy to construct in linear time.) This hypercube is our initial *box.* Let R denote the side length of this hypercube. A box is said to be *crowded* if it intersects more faces than some given threshold (at least as large as the incidence bound, β). If a box is crowded, then it is *split* by a hyperplane passing through the center of the box and perpendicular to its longest dimension. (Ties for the longest dimension can be broken arbitrarily, e.g., taking the lowest numbered axis.) The splitting process naturally defines a binary tree. The original box is the *parent* of the resulting boxes in this tree. The splitting process terminates when no remaining boxes are crowded. The resulting boxes, called *cells*, form a subdivision of space, each of which contains a constant number of obstacle boundary elements.

Observe that splitting will eventually terminate, because of the bounded incidence assumption. However, the number of cells in the subdivision cannot be bounded simply as a function of n.

To support segment shooting queries, we make one modification, which we call *smoothing.* (Smoothing is a well known operation on quad tree data structures and is better known under the names of *"restriction"* [11] or *"balancing"* [14, 4, 13]. We prefer the term "smoothing" because it avoids confusion with balancing in binary trees.) The idea of smoothing is quite simple. We say two cells are *neighbors* if they share a common boundary of dimension $d-1$. Define the *size* of a cell to be the length of its longest dimension. If there are two neighboring cells whose sizes differ by more than some constant factor (at least 2), then the larger cell is split. Splitting is repeated until all neighboring cells satisfy the size threshold. It has been shown by Moore[14] that smoothing does not alter the size of the subdivision except by a constant factor (in general depending on dimension).

To analyze the time for query processing, we first need to provide a relationship between the size of cells in the subdivision and strongly simple balls. For these results, we use the standard notion of simplicity (not C-simplicity).

Lemma 2.1 *Let $\epsilon > 0$, and let b be an ϵ-strongly simple ball of radius r. If b intersects a cell of the decomposition of size s, then $s \ge \alpha r$ for some constant α (depending on ϵ and dimension).*

Proof: Let $\alpha = \epsilon/(6\sqrt{d})$. For concreteness, assume that in smoothing, we split a cell if a neighbor is of size less than 1/2 of its size, and that $\epsilon \le 1$. Suppose to the contrary that there exists a cell that violates the conditions of this lemma. Consider the smallest such cell c, let s denote the size of c, and let p denote the parent box of c. Box p may have split for one of two reasons. Either (1) p itself is not simple, or (2) p is the neighbor of a cell c' whose size is less than 1/2 the size of p, and we split p for smoothing. The size of p is at most $2s$, and hence the diameter of p is at most $2s\sqrt{d}$.

In case (1), since c intersects b, then within distance $2s\sqrt{d}$ of the boundary of b there is a crowded cell. But,

$$2s\sqrt{d} < 2\alpha r\sqrt{d} = \frac{2\epsilon r\sqrt{d}}{6\sqrt{d}} \le \epsilon r.$$

Therefore, the $(1+\epsilon)$ expansion of b is not simple, a contradiction.

In case (2), let s' denote the size of c'. Since $s' < 2s/2 = s$, and since sizes vary by factors of 2, $s' \le s/2$. We claim that there is an ϵ-strongly simple ball b' of radius $r/2$ that contains c' (see Fig. 3). This will provide us with the desired contradiction, because c' satisfies the conditions of the lemma (since c was the smallest cell not satisfying the lemma and $s' < s$), and hence

$$\frac{s}{2} \ge s' \ge \alpha\frac{r}{2}$$

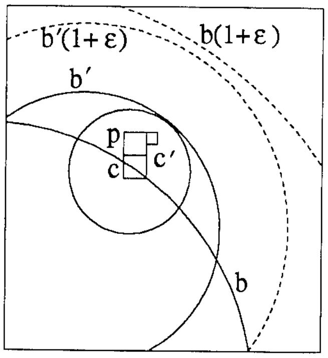


Figure 3: Proof of Lemma 2.1.

implying $s \geq \alpha r$, contradicting our earlier hypothesis.

To prove the existence of b', first observe that since the size of c' is at most $s/2$, its diameter is at most $s\sqrt{d}/2$. Since c' is a neighbor of p, and since p intersects b, c' is contained within a ball centered within b, whose radius is the sum of the diameters of p and c', which is less than $3s\sqrt{d}$. (This is the smallest circle in Fig. 3.) Since $s < \alpha r$ it is easy to verify that the radius of this ball is less than $r\epsilon/2$. Assuming $\epsilon \leq 1$, it is easy to see that there is a ball b' of radius $r/2$ that contains this ball, and is no further from boundary of b. Because b is within distance $3s\sqrt{d}$ of the boundary of b, the $(1+\epsilon)$ expansion of b' is at most a distance

$$\epsilon\frac{r}{2} + 3s\sqrt{d} < \epsilon\frac{r}{2} + 3\alpha r\sqrt{d} = \epsilon r$$

from the boundary of b. Thus, b' lies within the $(1+\epsilon)$ expansion of b, and so is simple. □

Lemma 2.2 *Given an ϵ-strongly simple ball b of radius r, the number of cells that intersect b in the subdivision is bounded above by a constant (depending on ϵ and dimension).*

Proof: By Lemma 2.1, the size of a cell intersecting b is bounded from below by αr. Thus, the number of such cells is bounded above by the number of squares of this side length that can be packed around (inside or touching) b, which is on the order of $1/\alpha^d$. □

Lemma 2.3 *Let q be a point in the enclosing hypercube. We can locate the cell containing q in time $O(\log(R/r))$, where r is the radius of the largest strongly simple ball containing q, and R is the side length of the bounding hypercube.*

Proof: The search for q is a simple descent through the PM *k-d* tree. With every d levels of descent, the size of the associated regions decreases by a factor of 1/2. Thus, if the search continues for k steps, then the search terminates at a region that contains q and whose size s obeys $s \leq \frac{R}{2^{\lfloor k/d \rfloor}}$. But Lemma 2.1 implies that $s \geq \alpha r$, for a constant α, so that $k = O(\log(R/r))$. □

Segment shooting queries are processed by first locating the cell containing the origin of the segment, and then walking along the segment, cell by cell, through the subdivision. Because neighboring cells differ in size by a constant amount, the number of neighbors of a given cell is at most a constant (dependent on d). By the fact that each cell contains a constant number of boundary elements, it follows that the time to perform this walking is proportional to the number of cells visited (with constant factors depending on d). Let $scc(s)$ denote the simple cover complexity of a segment s (more precisely, the complexity of the obstacle-free initial subsegment of s), and let B be a strongly simple cover for s. By Lemma 2.2, the number of subdivision cells intersecting each ball in B is a constant, and hence, since these balls cover s, the number of subdivision cells traversed by s is $O(|B|) = O(scc(s))$. By combining this with the previous lemma, it follows that segment shooting queries can be answered in $O((\log R/r) + scc(s))$ time. Observe that, by a similar argument, the number of cells in the subdivision is $O(scc(P))$, where $scc(P)$ denotes the simple cover complexity of the obstacle-free space. Since each cell contains only a constant number of boundary elements, the size of the final subdivision is $O(scc(P))$. From this we have the following analysis of the smoothed PM *k-d* tree subdivision.

Theorem 2.1 *Let P be a collection of polyhedral obstacles satisfying the bounded incidence assumption. There exists a data structure of size $O(scc(P))$ from which segment shooting queries can be answered in $O((\log R/r) + scc(s))$ time.*

At this point we remark that there is another way to achieve the same result without the use of smoothing. The purpose of smoothing the subdivision is to guarantee that the number of cells that are neighbors to any given cell is a constant. However, it is possible to avoid smoothing, and instead traverse the tree structure itself to locate neighboring cells (see [2], for example). It can be shown that finding neighboring cells by searching the tree does not increase the asymptotic running time of segment shooting queries. We have presented here the approach based on smoothing, since smoothing will be needed for the approach described in the next section.

This PM *k-d* tree-based data structure has the advantages of simplicity and ease of generalization to higher dimensions. Its principle disadvantages are that the space and, hence, preprocessing time bounds are dependent on the simple cover complexity of the obstacle set, rather than a function of n.

3 The Planar Case

In this section we modify the simple PM *k-d* tree construction given in the previous section to handle segment shooting for a set of polygonal obstacles, P, in the plane. The principle improvements are the following.

- Query time is based on C-complexity rather than standard simple cover complexity.
- Preprocessing time is $O(n \log n)$.
- Space is $O(n)$.

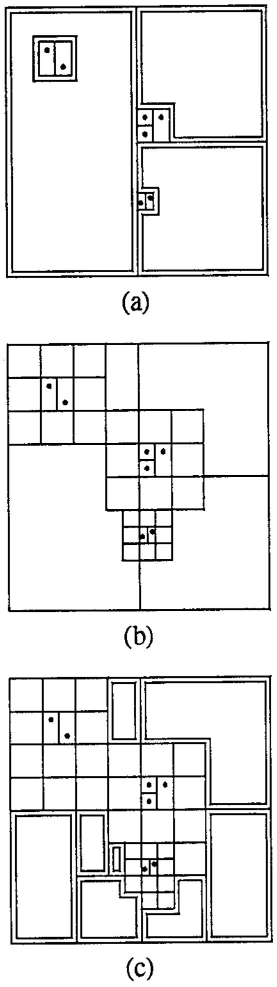
Interestingly, the subdivision presented here has a number of similarities to the quality mesh triangulations of [4, 13]. Since there are no restrictions on the geometric structure of our subdivision, we can provide the space and preprocessing time improvements listed above. (In mesh generation, the size of the triangulation depends on the geometric structure of the input.)

3.1 Box Decomposition

Preprocessing is based on a hierarchical subdivision similar to the PM *k-d* tree decomposition described in the last section, but in which a new operation called *shrinking* may be applied. The resulting structure is called a *box-decomposition tree.* This is a standard decomposition procedure for point sets, which can be found, for example, in [2, 3, 5, 9, 17]. As in the previous section, the subdivision partitions an enclosing square containing the obstacle set into a collection of cells, with each cell being bounded by a constant number of edges, which may be either segments of obstacle edges or axis aligned segments. We do not require that our subdivision be a cell complex, since there may be vertices lying in the interior of edges; however, the number of vertices in the interior of any edge will be a constant. Thus, we can easily augment such a subdivision into a cell complex or a triangulation with a linear number of additional Steiner points.

Each node of the box-decomposition tree corresponds to one of two types of objects called *enclosures*: (1) *boxes*, which are rectangles with the property that the ratio of the longest to shortest side is at most 2, and (2) *doughnuts*, which consist of the set theoretic difference of two boxes, an *inner box* contained within an *outer box.* The *size* of a box is defined to be the length of its longest side, and the *size* of a doughnut is the size of its outer box. As mentioned earlier, the subdivision construction begins by assuming that the set of obstacles are contained within a unit square. We begin by constructing a box-decomposition tree for the vertex set of the obstacles (ignoring obstacle edge). For completeness, we outline the procedure here.

Inductively, we assume that the vertex set to be decomposed is contained within a box. If there is only one vertex, then the decomposition terminates here. Otherwise, as with *k-d* trees, we split the box into two identical boxes by passing a horizontal or vertical line through its longest side, ***provided that there exists at least one vertex on either side of this splitting line.*** If so, we partition the vertices into these two boxes, and recursively subdivide each. This process is called *splitting.*


(a)

(b)

(c)

Figure 4: Box decomposition.

Without the provision above that there exists a vertex on each side of the splitting line, the splitting process could result in an arbitrarily long series of trivial splits, in which no vertices are separated from any other, and this in turn could result in a subdivision of size greater than $O(n)$. To avoid this, if there is *not* a vertex on either side of the splitting line, we perform a different decomposition step called *shrinking.* Shrinking consists of finding the smallest *k-d* tree box that contains all the vertices. By a *k-d* box, we mean any box that could be generated by some repeated application of the *k-d* splitting rule. This box can be computed in constant time assuming a model of computation where integer logarithm and floors can be computed in constant time. (This assumption can be overcome with a somewhat more flexible definition of boxes and shrinking.) This produces two enclosures. One is the bounding box containing the vertices, and the other is the surrounding doughnut that contains no vertices. We recursively subdivide the region lying within the bounding box. Note that after applying a shrinking step, the next step will be a splitting step. See Fig. 4(a) for an illustration of the decomposition.

By associating enclosures by containment, we naturally have a tree-structure, the box-decomposition tree, where each internal node has two children (either the left and right side from a split, or the doughnut and inner box from a shrink). The leaves of this tree define a subdivision of the plane into boxes and doughnuts. These are called *basic enclosures.*

3.2 Smoothing

To handle segment shooting by walking the ray through the subdivision, we will need to smooth the subdivision as in the previous section. The presence of shrinking requires that this be done more carefully than in the previous section, in order to keep the size of the resulting data structure from growing by more than a constant factor. The problem is that shrinking, by its very nature, introduces neighboring enclosures of significantly different sizes. Thus, the operations of shrinking and smoothing are incompatible in some sense. We overcome this by performing smoothing in two phases; the first phase handles inner boxes, and the second is the standard smoothing process.

The first phase of smoothing is to surround the inner box of each doughnut with a "buffer zone" of eight identical boxes (see Fig. 4(b)). If the inner box touches the boundary of the outer box, then these boxes may spill over into neighboring enclosures. In the case of spill-over, these boxes may already be a part of the decomposition, and no further action is needed. If spill-over occurs, and these boxes are not part of the decomposition, then this neighbor is decomposed by applying either splitting or shrinking appropriately, until we have produced a legal box-decomposition. This can be implemented by imagining that each of these newly created boxes contains a vertex of the obstacle set at its center, and then reapplying the box-decomposition algorithm described above (see Fig. 4(c)).

As before, we define a *neighbor* of a basic enclosure to be any other basic enclosure such that the two share a common line segment on their boundaries. However, for the purpose of the second phase of smoothing we make the following exception. Any enclosure which was created as part of the buffer zone for an inner box (and hence is empty) is not considered to have any larger enclosures as a neighbor. This exception is made for a couple of technical reasons. The most important is that we do not want the buffer zone around a small inner box to induce a much larger surrounding doughnut to split (or else we will undo the space savings achieved by shrinking). Henceforth, we assume the algorithm maintains, for each enclosure, a list of pointers to all larger neighboring enclosures. Note that there can be at most a constant number of such enclosures.

The second phase of smoothing is to determine whenever two neighboring enclosures differ in size by more than some constant factor (at least 2); in such a case, the larger enclosure is repeatedly split (never shrunk), until the difference in size falls below this factor. When splitting is applied to a shrinking node, the doughnut is split. Because we have chosen inner boxes to be *k-d* enclosures, it is not possible that the splitting line will pass through the inner box. Thus, after splitting a doughnut, there will be a new smaller doughnut with the same inner box, and a regular box (which will contain no vertices). If the doughnut shrinks so far that the outer box and inner box become equal, this trivial shrinking node can be deleted from the box-decomposition tree.

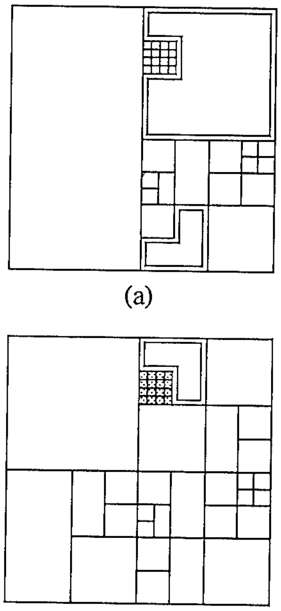


Figure 5: Smoothing: Second step.

This phase of the smoothing process is implemented in a bottom-up manner, working from small enclosures to larger enclosures. When a small enclosure identifies a sufficiently large neighboring enclosure, it splits the neighbor, creating a constant number of new enclosures. In Fig. 5(a) we show the output of the first smoothing phase, and in (b) we show the final output of the second smoothing phase. (Observe that the shaded enclosure in (b) did not induce the large enclosure to its left to be split, since the shaded enclosure is part of a buffer zone, and hence is not a neighbor of the larger enclosure.)

Lemma 3.1 *The smoothed box-decomposition tree can be constructed in $O(n \log n)$ time and $O(n)$ space.*

Proof: The proof that the initial box-decomposition construction can be performed in $O(n \log n)$ time and $O(n)$ space is well known (see, e.g. [2] or [5].) The first phase of smoothing is effectively equivalent to running a box-decomposition with at most $8n$ additional points, and so can be accomplished within the same time bound. We claim: (1) that the second smoothing phase can be performed in time proportional to the size of the final decomposition, and (2) that the final decomposition is of size $O(n)$. To prove (1) it suffices to observe that each splitting operation can be performed in $O(1)$ time and results in the creation of at least one new enclosure. The algorithm maintains the box-decomposition tree and, for each enclosure, it maintains the enclosure's size and a list of pointers to all larger neighboring enclosures (of which there can be only a constant number). With this information, each splitting operation can be performed in $O(1)$ time.

To establish (2) we apply a charging argument. As noted, the size of the box-decomposition tree for a set of n points is $O(n)$. Thus the number of nodes in the box-decomposition tree after the first phase is $O(n)$. Let N_1 denote this set of nodes, and let N_2 denote the nodes of the tree after the second phase. For each node in N_2 that is split we will charge this node to a node of N_1 in such a way that no node in N_1 is charged more than a constant number of times. This will complete the proof by establishing that the size of N_2 is $O(n)$.

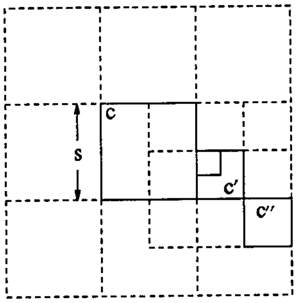


Figure 6: Charging scheme.

Let c be a node of N_2, and let s be the size of c. If the box which c represents is split as a result of smoothing, we claim that either c or at least one of the 8 identical boxes that surround c corresponds to a node which exists in N_1. We prove this by induction on the size of the enclosures. The reason that c split is either because c was already split in N_1 (in which case the claim is trivially true), or because there is a neighboring enclosure of size no greater than $s/4$. This neighbor cannot be part of a buffer zone, for otherwise it could not induce c to split. Therefore, the parent of this neighbor, denoted c', is an enclosure of size at most $s/2$ that is also a neighbor of c. By the induction hypothesis, the parent of c' or one of its 8 surrounding enclosures of size $s/2$ corresponds to a node in N_1 that was split. Call this node c''. Observe that since c'' is in a surrounding cell of the c', c'' must be contained within one of the 8 enclosures surrounding c (see Fig 6). We claim that this enclosure must be part of N_1. This can only fail to be true if c'' is enclosed directly within a much larger shrinking node. But this would imply that c'' is part of a buffer zone, and hence could not have induced c' to split.

For our charging scheme, we charge each node in N_2 that is split to a node of equal size in N_1. We have just established the existence of this node. Since each node in N_1 can be assessed charges only by itself or the nodes corresponding to one of its 8 surrounding enclosures, it follows that each node of N_1 is charged at most 9 times, and hence $|N_2| \leq 9|N_1|$. (For tighter bounds on the size of smoothed subdivisions, see [14].) □

Lemma 3.2 *After smoothing, the number of neighbors of any basic enclosure is* $O(1)$.

Proof: Smoothness guarantees that, except for the case of the enclosures of a buffer zone, neighboring enclosures are of roughly equal size. This implies that along any edge of any enclosure, there can be at most a constant number of neighboring enclosures. In the case of the enclosures of a buffer zone, observe that no matter how densely points are packed into the inner box, the propagation of splitting can result in the outer edges of the buffer zone being split at most once. This is illustrated in Fig. 7. □

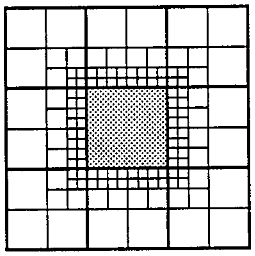


Figure 7: Propagation of smoothing.

Due to the doughnuts, basic enclosures may not be convex; indeed, the vertices of an inner box of a doughnut are reflex vertices. We can make the overall subdivision convex by extending the left and right (vertical) sides of each inner box until they contact the boundary of the outer box. This subdivides each doughnut into four rectangles. Observe that this increases the stabbing number of any directed segment by at most a constant factor, and so has no significant effect on the complexity arguments to be made later.

3.3 Adding Obstacle Edges

To complete the description of the decomposition algorithm, we introduce the obstacle edges. The problem with simply adding the obstacle edges to the box-decomposition is that the number of intersections between obstacle edges and box-decomposition edges may be much larger than $O(n)$. To guarantee that the number of intersections is linear in n, we apply a simple trimming procedure, which trims each edge of the smoothed box-decomposition subdivision at its first and last intersection with an obstacle boundary. Let us assume that both the obstacle set and box-decomposition have been preprocessed (by standard means—e.g., trapezoidization, and a point location data structure) to support horizontal and vertical ray-shooting queries in $O(\log n)$ time. Intuitively, one can imagine each vertex of the box-decomposition subdivision firing a bullet along each of its incident horizontal and vertical edges, until either reaching the end of this edge, or until hitting an obstacle. The edges are then trimmed so only these bullet-path subedges remain.

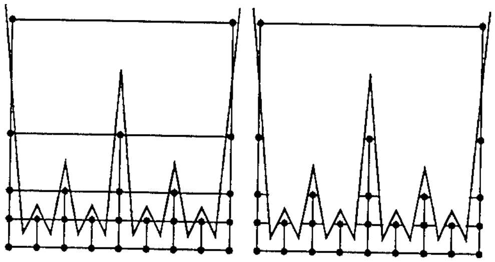
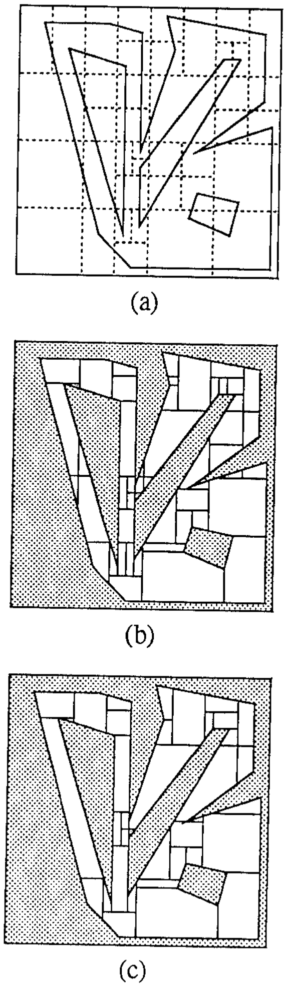


Figure 8: The problem with simple trimming.

Directly applying the simple trimming procedure described above can result in regions of complexity $\Omega(n)$ (see Fig. 8). To deal with this, we isolate obstacle vertices within the subdivision. Before doing any trimming, for each obstacle vertex v we determine its associated enclosure, c, in the smoothed box decomposition (this is given to us by the decomposition). We join v to the boundary of c by at most 4 vertical/horizontal segments, omitting any such connection that penetrates the obstacle containing v; this results in at most 4 new (subdivision) vertices. This modification of the box-decomposition subdivision can be performed for all n vertices in total time $O(n)$, while increasing the size of the subdivision by only a constant factor.

Now we apply the above-mentioned trimming procedure for each edge of the modified box-decomposition subdivision. Since this involves shooting $O(n)$ directed (horizontal/vertical) segments, the entire process takes $O(n \log n)$ time. Observe that the resulting subdivision has total size $O(n)$, since we have created at most two new edges for each of the $O(n)$ edges in the box-decomposition subdivision. Within this same time bound we can convert this collection of segments (which includes box-decomposition edges as well as obstacle edges) into a planar subdivision data structure. The 2-faces of the resulting subdivision, which are necessarily convex, are called *cells*. We say that two cells are *neighbors* if they share a common nonobstacle (box-decomposition) edge. An example of this is shown in Fig. 9, where (a) shows the initial smooth box-decomposition of the point set, and (b) shows the result of trimming. (In connecting obstacle vertices to the sides of the subdivision, we have used as few segments as needed to resolve the reflex angles.)

Notice that the resulting subdivision inherits one essential feature of the smoothed box-decomposition, namely, that each cell in the subdivision has at most a constant number of neighbors. Also observe that each cell of the resulting subdivision is convex and is bounded by at most 6 sides (two obstacle edges and the four sides of a bounding rectangle).



(a)

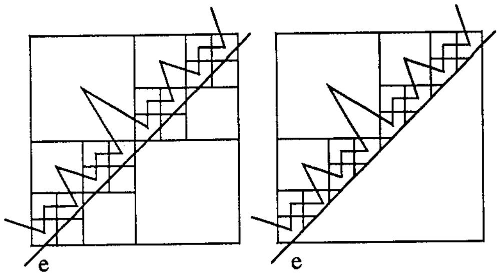
(b)

(c)

Figure 9: The final trimming procedure.

3.4 Merging

If we stop at this point, we claim that the subdivision constructed so far has the property that any segment s stabs as many cells of the subdivision as the simple cover complexity of s. However, we want to strengthen this to C-simple cover complexity. This is generally not true, since it may contain cells that are smaller than they should be. It is important that, for every point p in the plane, the size of the cell containing p is at least a constant factor of the radius of the largest C-simple ball that contains p. (This is the C-simple version of Lemma 2.1.) This may be violated because the placement of box-decomposition vertices was made without knowledge of the underlying structure of obstacle edges. An example is shown in Fig. 10(a). A large number of small subdivision cells lie to the right of edge e in this figure. The existence of these cells is justified from the vertices lying to the left of e. However, a ray lying just to the right of e and parallel to e will have a very low C-simple cover complexity because these other vertices are not reachable from the ray, and hence we cannot afford to traverse these small cells.



(a) (b)

Figure 10: Merging.

We apply the following merging process to eliminate these unnecessary regions. We traverse the enclosures of the box-decomposition subdivision, in a bottom-up fashion (starting at the bottom of the box-decomposition tree) visiting enclosures in increasing order of size. Consider a pair of neighboring cells that were siblings in the decomposition process (in the sense that they were once part of a parent node that was split or shrunken). If their union satisfies the following properties:

(1) it has at most two obstacle edges on its boundary,

(2) it does not have more than a constant number c neighboring cells (where $c \geq 8$), and

(3) it is convex,

then merge these two cells into one cell (effectively undoing the decomposition performed by the decomposition algorithm). Repeat this until no sibling cells remain that can be merged. After this merging process has been finished, observe that cells still satisfy the properties of bounded complexity, bounded number of neighbors, and convexity. The outcome of this merging procedure is shown in Fig. 10(b).

Summarizing the discussion of this section, we have the following result.

Lemma 3.3 *Given a collection of polygonal obstacles P in the plane with n total vertices, the size of the final subdivision is $O(n)$, and the subdivision can be constructed in $O(n \log n)$ time. The cells of this subdivision satisfy properties (1)-(3) above.*

At this point we can present the result analogous to Lemma 2.1. Recall that in order to make cells convex, we introduced new edges incident to the vertices of inner boxes (in Section 3.2) and to obstacle vertices (in Section 3.3). The effect of adding these edges was to subdivide each enclosure of the smoothed box decomposition into a constant number of convex subcells. Since each enclosure of the box decomposition can contain at most a constant number of nonreflex vertices (as a consequence of box decomposition), the creation of these subcells does not affect the number of cells traversed in query processing by more than a constant factor. Thus, we can define the *size* of such subcells to be the size of original enclosures.

Lemma 3.4 *For every point p in the plane, the size of the cell containing p in the subdivision is at least a constant factor times the radius of the largest C-simple ball that contains p.*

Proof: Let r denote the radius of the largest C-simple ball that contains p. Let s be the size of the cell c containing p. Let c' be the sibling of c in the subdivision (or in general, the next neighboring cell that would be considered for merging with c). The cells c and c' were not merged because of a violation of one of the conditions (1), (2), or (3) above.

In the case of (1), the merger of c and c' would result in a cell having 3 or more reachable vertices within distance $s2\sqrt{2}$. Thus the ball of radius $s2\sqrt{2}$ centered at p is not C-simple. This implies that any C-simple ball that covers p can be of radius at most $s2\sqrt{2}$, and hence $s \geq r/(2\sqrt{2})$.

In the case of (2), the proof is by induction on the size of cells. If c' was separated from c by splitting (as opposed to shrinking), then c and c' are of equal size. The fact that condition (2) is violated implies that there is a reachable cell c'' neighboring either c or c'' that is no larger than $s/2$ (since otherwise there could be at most 8 neighbors of their union). Applying the induction hypothesis to the closest point p' to p in c'', implies that the largest C-simple ball containing p' is of size $\alpha s/2$, for some constant α. Note that p' cannot be further than $s2\sqrt{2}$ from p, and so any C-simple ball that covers p can be of radius at most $s(2\sqrt{2} + \alpha/2)$. Choosing any

$\alpha \geq 4\sqrt{2}$ will satisfy the hypothesis. If c and c' were separated by shrinking, then one of these cells is a doughnut and the other is an inner box. If c is the doughnut, then c' is smaller than c, and the same induction argument can be applied. We claim that c cannot be the inner box, since any neighbor of the neighbors of an inner box are a subset of the neighbors of the doughnut. Thus, merging c and c' could not violate condition (2).

Finally, in the case of condition (3), the edge separating c and c' has been added simply to resolve a reflex vertex. We can simply apply the arguments given in cases (1) or (2) to the larger enclosure of size s in the box decomposition that contains this vertex. □

The analogue to Lemma 2.2 follows from exactly the same argument as before.

Lemma 3.5 *Given a C-simple ball b of radius r, the number of cells that intersect any connected component of b in the subdivision is bounded above by a constant.*

3.5 Query Processing

As we mentioned earlier, query processing involves two steps. Given a directed segment $\overline{pq}$, we first apply any point location algorithm to determine the location of the starting point p in $O(\log n)$ time. Once the cell containing p has been located, the algorithm simply walks from one cell to the next. This can be done in time proportional to the number of cells traversed, since each cell is bounded by a constant number of edges (at most two obstacle edges and at most four horizontal and vertical edges of the associated (trimmed) enclosure), and each cell has a constant number of neighbors.

To analyze the running time of query processing it suffices to count the number of cells visited. Let $scc(s)$ denote the C-simple cover complexity of a segment s (or actually the complexity of the obstacle free initial subsegment of s), and let B be a strongly C-simple cover for s. By Lemma 3.5, the number of subdivision cells intersecting each ball in B is a constant, and hence, since these balls cover s, the number of subdivision cells traversed by s is $O(|B|) = O(scc(s))$.

Theorem 3.1 *Let P be a collection of polygonal obstacles in the plane (satisfying the bounded incidence assumption). Let n denote the number of vertices in P. In time $O(n \log n)$ a data structure of size $O(n)$ can be constructed, from which segment shooting queries can be answered in $O(\log n + scc(s))$ time.*

References

[1] P. K. Agarwal. A deterministic algorithm for partitioning arrangements of lines and its applications. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 11–22, 1989.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu. An optimal algorithm for approximate nearest neighbor searching. In *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, 1994, 573–582.

[3] M. Bern. Approximate closest-point queries in high dimensions. *Inform. Process. Lett.*, 45:95–99, 1993.

[4] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. In *Proc. 31st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 231–241, 1990.

[5] P. B. Callahan and S. R. Kosaraju. A decomposition of multi-dimensional point-sets with applications to k-nearest-neighbors and n-body potential fields. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 546–556, 1992.

[6] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. In *Proc. 18th Internat. Colloq. Automata Lang. Program.*, volume 510 of *Lecture Notes in Computer Science*, pages 661–673. Springer-Verlag, 1991.

[7] B. Chazelle and L. J. Guibas. Visibility and intersection problems in plane geometry. In *Proc. 1st Annu. ACM Sympos. Comput. Geom.*, pages 135–146, 1985.

[8] S. W. Cheng and R. Janardan. Space-efficient ray shooting and intersection searching: algorithms, dynamization and applications. In *Proc. 2nd ACM-SIAM Sympos. Discrete Algorithms*, pages 7–16, 1991.

[9] K. L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17:830–847, 1988.

[10] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 54–63, 1993.

[11] B. Von Herzen and A. H. Barr. Accurate triangulations of deformed, intersecting surfaces. *Computer Graphics*, 21(4):103–110, July 1987.

[12] J.S.B. Mitchell. Computational geometry methods for virtual reality. Manuscript, Applied Math, SUNY Stony Brook, 1993.

[13] S. A. Mitchell and S. A. Vavasis. Quality mesh generation in three dimensions. In *Proc. 8th Annu. ACM Sympos. Comput. Geom.*, pages 212–221, 1992.

[14] D. W. Moore. *Simplicial mesh generation with applications.* Ph.D. thesis, Dept. Comput. Sci., Cornell Univ., Ithaca, NY, 1992. Report 92-1322.

[15] M. Overmars, H. Schipper, and M. Sharir. Storing line segments in partition trees. *BIT*, 30:385–403, 1990.

[16] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.

[17] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.

Efficient Algorithms for Generalized Intersection Searching on Non-Iso-Oriented Objects

Prosenjit Gupta* Ravi Janardan* Michiel Smid†

Abstract

Generalized intersection searching problems are a class of geometric query-retrieval problems where the questions of interest concern the intersection of a query object with aggregates of geometric objects (rather than with individual objects.) This class contains, as a special case, the well-studied class of standard intersection searching problems and is rich in applications. Unfortunately, the solutions known for the standard problems do not yield efficient solutions to the generalized problems. Recently, efficient solutions have been given for generalized problems where the input and query objects are iso-oriented (i.e., axes-parallel) or where the aggregates satisfy additional properties (e.g., connectedness). In this paper, efficient algorithms are given for several generalized problems involving non-iso-oriented objects. These problems include: generalized halfspace range searching, segment intersection searching, triangle stabbing, and triangle range searching. The techniques used include: computing suitable sparse representations of the input, persistent data structures, and filtering search.

Keywords: Computational geometry, data structures, filtering search, geometric duality, intersection searching, persistence.

1 Introduction

Consider the following generic searching problem: Suppose that we are given a set S of n geometric objects in $\mathcal{R}^d$. Moreover assume that the objects come aggregated in disjoint groups, where the grouping is dictated by the underlying application. (The number of groups can range from 1 to $\Theta(n)$.) Our goal is to preprocess S into a data structure so that given any query object q, we can report or count efficiently the groups that are intersected by q. (We say that q *intersects a group* iff q intersects some object in the group.) Notice that we are not interested in reporting or counting the individual objects intersected by q as is the case in a *standard intersection searching* problem. Indeed the standard problem is a special case of the above formulation, where each group has cardinality 1. For this reason, we call our version a *generalized intersection searching* problem.

For our purposes, it will be convenient to associate with each group a different color and imagine that all the objects in the group have that color. Suppose that q intersects i groups. Then, we can restate our problem as: "*Preprocess a set S of n colored geometric objects so that for any query object q, the i distinct colors of the objects that are intersected by q can be reported or counted effi-*

*Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, U.S.A. Email: {pgupta, janardan}@cs.umn.edu. The research of these authors was supported in part by NSF grant CCR-92-00270. Portions of this work were done while RJ was visiting MS at the Max-Planck-Institut für Informatik, in Saarbrücken, Germany, in July 1993. RJ would like to thank the MPI for its generous support.

†Max-Planck-Institut für Informatik, D-66123 Saarbrücken, Germany. Email: michiel@mpi-sb.mpg.de. This author was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II).

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

ciently." This is the version that we will consider throughout the paper.

Before going further, let us illustrate the usefulness of our generalized formulation with some applications: **(1)** In designing a VLSI chip, the wires (line segments) can be grouped naturally according to the circuits they belong to. A problem of interest to the designer is determining which circuits (rather than wires) become connected when a new wire is to be added. This is an instance of the *generalized segment intersection searching* problem. **(2)** Consider a collection of supply points (e.g., warehouses) of different types in $\mathcal{R}^2$. We would like to preprocess these points so that given a demand point q and a radius r, we can determine the types of supply points that are within distance r from q. By using a well-known "lifting" transformation, this problem can be transformed to an instance of *generalized halfspace range searching* in $\mathcal{R}^3$.

1.1 Potential approaches

One approach to solving a generalized problem is to take advantage of known solutions for the corresponding standard problem. For example, to solve a generalized reporting problem, we can determine all the objects intersected by q (a standard problem) and then read off the distinct colors among these. However, with this approach the query time can be very high since q could intersect $\Omega(n)$ objects but only $O(1)$ distinct colors. Thus, the challenge in the generalized reporting problem is to attain a query time that is sensitive to the output size i, typically of the form $O(f(n) + i)$ or $O(f(n)+i \cdot \text{polylog}(n))$, where $f(n)$ is "small" (e.g., polylog(n) or n^p, where $0 < p < 1$). For a generalized counting problem, it is not even clear how one can use the solution for the corresponding standard problem (a mere count) to determine how many distinct colors are intersected. Nevertheless, we seek here query times of the form $O(f(n))$. Of course, in both cases, we seek solutions that are also as space-efficient as possible.

1.2 Previous work

While the standard problems have been investigated extensively, their generalized counterparts have been less studied. The generalized problems were first considered in [JL93], where efficient solutions were given for several problems defined on *iso-oriented objects* (i.e., the input and the query objects are axes-parallel). In [GJS93a], efficient solutions were given for the counting, reporting, and dynamic versions of several iso-oriented problems. In [GJS93b], solutions were given for generalized problems involving circular and circle-like objects (among other results). In [AvK93], Agarwal and van Kreveld consider the problem of reporting the intersections of a query line segment with color classes consisting of line segments and satisfying the property that each color class is a simple polygon or a connected component.

1.3 Summary of results and techniques

In this paper, we present efficient solutions to several generalized intersection searching problems that are defined on non-iso-oriented objects. Specifically, we consider the following problems: generalized halfspace range searching in $\mathcal{R}^d$, for any fixed $d \geq 2$, generalized segment intersection searching, triangle stabbing, and triangle range searching in $\mathcal{R}^2$. Our main results are summarized in Table 1. No results were known previously for any of these problems, with the exception of generalized segment intersection searching: For this problem, the following results were known: (a) $O(n^{1+\epsilon})$ space and $O(n^{1/2+\epsilon} + i)$ query time for color classes consisting of simple polygons or connected components [AvK93]; (b) $O((n + \chi)^2 \log n)$ (resp. $O((n + \chi)^2)$) space and $O(\log n + i)$ (resp. $O(\log^2 n + i)$) query time for general color classes [JL93]; and (c) $O(n^{1+\epsilon})$ space and $O((i + 1)\sqrt{n} \log^{O(1)} n)$ query time for general color classes [AvK93]. (χ is the number of pairwise-intersecting segments.)

Our results are based on a combination of several techniques: (1) Computing for each color class a sparse representation which captures essential information about the color class and allows us to reduce the generalized problem at hand to a standard problem. (2) The persistence-addition technique of Driscoll et al. [DSST89], which allows us to reduce a generalized problem to a generalized dynamic problem one dimension lower. (3) A version of filtering search which, in combination with

#	*Input objects*	*Query object*	*Space*	*Query time*
1	Points in $\mathcal{R}^d$	Halfspace in $\mathcal{R}^d$	$d = 2$ $n \log n$	$\log^2 n + i$ $n^{1/2}$
			$n \log^2 n$ $d = 3$ $n^{2+\epsilon}$ $n \log n$	$n^{1/2+\epsilon} + i$ $\log^2 n + i$ $n^{2/3+\epsilon}$
			$d \geq 2$ $n^{\lfloor d/2 \rfloor + \epsilon}$	$\log n + i \log^2 n$
2	Line segs. in $\mathcal{R}^2$	Halfplane	$n \log n$	$\log^2 n + i$ $n^{1/2}$
		Vertical ray	$n\alpha(n) \log n$	$\log^2 n + i$ $(n\alpha(n))^{1/2}$
3	Line segs. of length $\geq$ a constant in unit square	Line	$n \log n$	$\log^2 n + i$
4	Lines in $\mathcal{R}^2$	Vertical line segment	$n^{2.5-\mu} \log n$ $n^2 \log n$	$n^\mu + i$ $\log n + i$
5	Line segs. in $\mathcal{R}^2$	Vertical line segment	$(n + \chi) \log n$	$\log n + i$
		Arbitrary line segment		$\log^2 n + i \log n$
6	Triangles	Point	$n^{3/2} \log n$	$\log^2 n + i$
	Fat-Wedges		$n \log n$	$\log^2 n + i$
7	Points	Fat-Triangle	$n \log^3 n$	$\log^4 n + i \log^2 n$

Table 1: Summary of main results for generalized intersection searching problems; additional results can be found in the text. Wherever the output size, i, is missing in a query time bound, it is a counting problem. A *fat-wedge* or *fat-triangle* is one where each interior angle is greater than or equal to a fixed constant. Here:

n	:	input size
i	:	output size (number of distinct colors intersected)
ϵ	:	arbitrarily small constant > 0
μ	:	tunable parameter, $0.5 < \mu < 1$
χ	:	number of pairwise-intersecting segments, $0 \leq \chi \leq \binom{n}{2}$
$\alpha(n)$	:	slow-growing inverse of Ackermann's function

persistence, yields a space-query time tradeoff. Moreover, when the input objects or query objects satisfy certain reasonable conditions (e.g., fatness), then we use further ideas to obtain very efficient solutions. Due to space limitations, we will describe only a subset of our results here and will omit all proofs and many details. The full paper is available as [GJS93c].

2 Generalized halfspace range searching in $\mathcal{R}^d$

Let S be a set of n colored points in $\mathcal{R}^d$, for any fixed $d \geq 2$. We show how to preprocess S so that for any query hyperplane Q, the i distinct colors of the points lying in the halfspace Q^- (i.e., below Q) can be reported or counted efficiently. We first give solutions for $\mathcal{R}^2$ and $\mathcal{R}^3$. Let $\mathcal{F}$ denote the well-known point-hyperplane duality transform [Ede87]. Using $\mathcal{F}$ we map S to a set S' of hyperplanes (lines in $\mathcal{R}^2$ and planes in $\mathcal{R}^3$) and map Q to the point $q = \mathcal{F}(Q)$. Our problem is now equivalent to: "Report or count the i distinct colors of the hyperplanes lying on or above q, i.e., the hyperplanes that are intersected by the vertical ray r emanating upwards from q."

Let S_c be the set of hyperplanes of color c. For each color c, we compute the *upper envelope* E_c of the hyperplanes in S_c. In $\mathcal{R}^2$, E_c is an unbounded convex chain, and in $\mathcal{R}^3$, E_c is an unbounded convex polytope whose facets are convex polygons. It is clear that (i) r intersects a c-colored hyperplane iff r intersects E_c and, moreover, (ii) if r intersects E_c, then r intersects the interior of a unique facet of E_c. (For this version of the paper, we assume that r does not intersect two or more facets of E_c at a common boundary; in the full paper [GJS93c], we show how to remove this assumption.) Let $\mathcal{E}$ be the collection of the envelopes of the different colors. By the above discussion, our problem is equivalent to: "Report or count the facets of $\mathcal{E}$ that are intersected by r", which is a standard intersection searching problem! Since we use different techniques to solve this problem in $\mathcal{R}^2$ and in $\mathcal{R}^3$, we will discuss each case separately.

2.1 Solving the ray–envelope intersection problem in $\mathcal{R}^2$

We store the x-projections of the line segments of $\mathcal{E}$ in a *segment tree* T. Let v be any node of T. Associated with v is an x-interval $I(v)$. Let *Strip*(v) be the vertical strip defined by $I(v)$. We say that a segment $s \in \mathcal{E}$ is *allocated* to a node $v \in T$ iff $I(v) \neq \emptyset$ and s crosses *Strip*(v) but not *Strip*$(parent(v))$. Let $\mathcal{E}(v)$ be the set of segments allocated to v. Within *Strip*(v), the segments of $\mathcal{E}(v)$ can be viewed as lines since they cross *Strip*(v) completely. Let $\mathcal{E}'(v)$ be the set of points dual to these lines. We store $\mathcal{E}'(v)$ in an instance $H(v)$ of the standard halfplane reporting (resp. counting) structure for $\mathcal{R}^2$ given in [CGL85] (resp. [Mat92b]). This structure has size $O(m)$ and a query time of $O(\log m + k_v)$ (resp. $O(m^{1/2})$), where $m = |\mathcal{E}(v)|$ and k_v is the output size at v.

To answer a query, we search in T using q's x-coordinate. At each node v visited, we need to report or count the lines intersected by r. But, by duality, this is equivalent to answering, in $\mathcal{R}^2$, a halfplane query at v using the query $\mathcal{F}(q)^- = Q^-$, which we do using $H(v)$.

Theorem 2.1 *A set S of n colored points in $\mathcal{R}^2$ can be stored in a data structure of size $O(n \log n)$ so that the i distinct colors of the points lying in any query halfplane can be reported (resp. counted) in time $O(\log^2 n + i)$ (resp. $O(n^{1/2})$).* □

Using a similar approach, we can also solve Problem 2 in Table 1.

2.2 Solving the ray–envelope intersection problem in $\mathcal{R}^3$

We triangulate the facets of E_c for all colors c. For any triangle t, let $h(t)$ be its supporting plane. Let t' and q' be the projections of t and q (the origin of ray r) on the xy-plane respectively. Clearly, t is intersected by r iff (a) t''s interior contains q' and (b) $h(t)$ is on or above q.

Wlog assume that t' has a vertical side; otherwise decompose it into two such triangles. To find the triangles satisfying condition (a), we store each t' in a segment tree T according to its x-span. Let v be any node of T and let $A(v)$ be the set of triangles allocated to v. Let $m = |A(v)|$. Note that if $t' \in A(v)$ then both its non-vertical sides, call the upper one t'_u and the lower one t'_l, cross *Strip*(v). If $q' \in$ *Strip*(v), then q' is in t''s interior iff q' is above t'_l and below t'_u. Since t'_l and t'_u behave like lines within *Strip*(v), by duality we have that $q' \in t'$ iff in $\mathcal{R}^2$ one endpoint of $\overline{\mathcal{F}(t'_l)\mathcal{F}(t'_u)}$ lies in the open halfplane $\mathcal{F}(q')^-$ and the other lies in the open halfplane $\mathcal{F}(q')^+$. Next, consider condition (b). By duality, $h(t)$ is on or above q iff in $\mathcal{R}^3$ the point $\mathcal{F}(h(t))$ is in the halfspace $\mathcal{F}(q)^-$.

So, our problem at v is to report or count the i_v triangles of $A(v)$ that satisfy the above halfplane and halfspace queries. We can do this by augmenting v with a 3-level data structure $\mathcal{D}(v)$, using partition trees [Mat92a] in the outer two levels of $\mathcal{D}(v)$ and the data structure of [AHL90] for halfspace range reporting at the innermost level. For the counting problem, we use partition trees at all three levels. $\mathcal{D}(v)$ can be built in space

$O(m \log m)$ (resp. $O(m)$) so that all the desired triangles can be reported (resp. counted) in time $O(m^{1/2+\epsilon} + i_v)$ (resp. $O(m^{2/3+\epsilon})$). For the reporting problem, instead of partition trees, we can also use 2-dimensional cutting trees [Mat91a] for the two outer levels of $\mathcal{D}(v)$. Then $\mathcal{D}(v)$ has size $O(m^{2+\epsilon})$ and query time $O(\log m + i_v)$. Plugging these bounds into the segment tree, we get:

Theorem 2.2 *The reporting version of the generalized halfspace range searching problem for a set of n colored points in $\mathcal{R}^3$ can be solved in $O(n \log^2 n)$ (resp. $O(n^{2+\epsilon})$) space and $O(n^{1/2+\epsilon} + i)$ (resp. $O(\log^2 n + i)$) query time, where i is the output size and $\epsilon > 0$ is an arbitrarily small constant. The counting version is solvable in $O(n \log n)$ space and $O(n^{2/3+\epsilon})$ query time.* □

2.3 Generalized halfspace range reporting in $d \geq 2$ dimensions

The preceding approach extends to $\mathcal{R}^d$, $d > 3$, but the space bound is high, namely, $O(n^{d\lfloor d/2 \rfloor+\epsilon})$. We now give an approach for the *reporting* problem in $\mathcal{R}^d$ ($d \geq 2$) that needs much less space. In preprocessing, we store the distinct colors in the input point-set S at the leaves of a balanced binary tree CT (in no particular order). For any node v of CT, let $C(v)$ be the colors stored in the leaves of v's subtree and let $S(v)$ be the points of S colored with the colors in $C(v)$. At v, we store a data structure $HSE(v)$ to solve the *halfspace emptiness* problem on $S(v)$, i.e., "does a query halfspace contain any points of $S(v)$?" $HSE(v)$ returns "true" iff the query halfspace is empty. If $|S_v| = n_v$, then $HSE(v)$ uses $O(n_v^{\lfloor d/2 \rfloor+\epsilon})$ space and has query time $O(\log n_v)$ [Mul93, page 290].

To answer a generalized halfspace reporting query we do a depth-first search in CT and query $HSE(v)$ at each node v visited. If v is a non-leaf then we continue searching below v iff the query returns "false"; if v is a leaf, then we output the color stored there iff the query returns "false".

Theorem 2.3 *For any fixed $d \geq 2$, a set S of n colored points in $\mathcal{R}^d$ can be stored in a structure of size $O(n^{\lfloor d/2 \rfloor+\epsilon})$ such that the i distinct colors of the points contained in a query halfspace Q^- can be reported in time $O(\log n + i \log^2 n)$. Here $\epsilon > 0$ is an arbitrarily small constant.* □

We note that the reporting (resp. counting) problem can also be solved in $O(n^{d+\epsilon})$ space and $O(\log n + i)$ (resp. $O(\log n)$) time, using constant-depth cutting trees.

3 Generalized intersection reporting on lines and line segments

We consider several versions of the generalized intersection searching problem for a set S of n colored lines or line segments; the query q is a line or a line segment.

3.1 Querying colored line segments with a line

We dualize the colored line segments of S to a set S' of colored doublewedges and map the query line q to a point q'. Thus, our problem reduces to reporting the i distinct colors that are stabbed by q'. In Section 4, we solve this problem in $O(n^{3/2} \log n)$ space with a query time of $O(\log^2 n + i)$. In the rest of this section, we consider the case where the segments of S all lie in the unit square $\mathcal{U}$ and each segment has length at least λ, where $\lambda > 0$ is a constant. These assumptions are reasonable for practical applications and they allow a very efficient solution.

For now assume that all the segments intersect the y-axis Y. Thus, one endpoint of s has negative x-coordinate and the other has positive x-coordinate. Thus in the corresponding dual doublewedge one of the bounding lines has positive slope and the other has negative slope. We split each doublewedge into a *left-wedge* and a *right-wedge.* Note that each wedge is y-monotone. Consider the right-wedges. Because of y-monotonicity, q' is contained in a right-wedge w iff the horizontal, leftward-directed ray r emanating from q' intersects the boundary of w. Thus for each color c, we compute the *left-envelope* of the boundaries of all c-colored right wedges. If there are n_c c-colored right wedges, then the c-colored left-envelope has size $O(n_c)$ (see [Ede87, page 357, Problem 15.6]). In this way, we obtain a collection S'' of colored line segments. Note that (i) r intersects the boundary of a c-colored right-wedge iff r intersects a c-colored left-envelope and (ii) if r intersects a c-colored left-envelope then it intersects a unique line segment of this envelope. Thus we have a standard problem which we can solve in $O(n \log n)$ space and $O(\log^2 n + i)$ query time by using a segment tree as in Section 2.1.

What if the segments of S do not all intersect Y? Suppose that there is a constant K such that each segment intersects one of K fixed lines $Y_1, \ldots, Y_K$. Let $S_i \subseteq S$ be the set of segments intersecting Y_i. (If a segment intersects more than one Y_i, we put it in any one of the S_i's; thus the S_i's partition S.) For $1 \leq i \leq K$, we create a coordinate system $\mathcal{C}_i$, where Y_i is the y-axis and any line perpendicular to Y_i is taken as the x-axis. We give the segments of S_i coordinates in $\mathcal{C}_i$ and store them in the data structure described above. To answer a query, we query each of the K structures separately. The space and query time are as above.

We can now solve the problem stated at the beginning. Wlog assume that the origin is at the bottom-left corner of $\mathcal{U}$. Consider the $K = 2 + 2\lceil\sqrt{2}/\lambda\rceil$ lines $x = i \cdot \lambda/\sqrt{2}$ and $y = i \cdot \lambda/\sqrt{2}$, where $0 \leq i \leq \lceil\sqrt{2}/\lambda\rceil$. Since each segment has length at least λ, either its x-span or its y-span is at least $\lambda/\sqrt{2}$. Thus each segment intersects one of the K lines. We now use the above structure.

Theorem 3.1 *Let $\lambda > 0$ be a constant and let $\mathcal{U}$ be the unit square. A set S of n colored line segments in $\mathcal{R}^2$, where each segment has length at least λ and all segments lie in $\mathcal{U}$, can be stored in a structure of size $O(n \log n)$ such that the i distinct colors of the segments that are intersected by a query line q can be reported in time $O(\log^2 n + i)$.* □

3.2 Querying colored lines with a vertical line segment

We give a simple approach based on persistence [DSST89], and then show how to incorporate filtering search to get a space-query time tradeoff.

We divide the plane into $t = O(n^2)$ strips by drawing vertical lines through the intersection points of the n lines. Within any strip, V_k, the lines are totally ordered from top to bottom, as $E_k : \ell_1, \ell_2, \ldots, \ell_n$. Suppose that the vertical query segment q is in V_k and let ℓ_a and ℓ_b be the highest and lowest lines of E_k intersected by q. Our problem is now equivalent to the following *generalized 1-dimensional range searching* problem: Given colored integers $1, 2, \ldots, n$ (where integer j gets the color of ℓ_j), report the distinct colors in the query interval $[a, b]$. In [GJS93a], this problem is solved using a structure D_k of size $O(n)$ and a query time $O(\log n + i)$. D_k supports updates in $O(\log n)$ time with $O(\log n)$ memory modifications. By sweeping over the strips V_k, $1 \leq k \leq t$, and making the associated D_k's partially persistent, we get a solution with space $O(n^2 \log n)$ and query time $O(\log n + i)$.

In a nutshell, the idea for the space-query-time tradeoff is as follows: We extract from the sequence $\mathcal{E} = (E_1, \ldots, E_{t+1})$ a smaller subsequence $\mathcal{E}' = (E'_1, \ldots, E'_m)$ such that (i) E'_j and E'_{j+1} differ in just two (not necessarily adjacent) positions, (ii) for each $E_k \in \mathcal{E}$ there is an $E'_j \in \mathcal{E}'$ which "approximates" E_i in a sense that we will elaborate upon later, and (iii) $m = O(n^{2.5-\mu})$, where $0.5 < \mu < 1$ is a tunable parameter. Properties (i) and (iii) suggest that we can apply persistence to $\mathcal{E}'$ and get a scheme with space bound $o(n^2)$; property (ii) suggests that instead of querying E_k, as we might in the simple scheme, we can query E'_j in the new scheme and still be assured of correctness.

Formally, we define a sequence $b_1, b_2, \ldots, b_B$ of distinguished list positions called *borders*, where $b_i = (i-1)\lfloor n^\mu + 1\rfloor + 1$ and $B = \Theta(n^{1-\mu})$. We construct $\mathcal{E}'$ by scanning $\mathcal{E}$. Let E_i be the currently scanned list of $\mathcal{E}$. Let E'_j be the most recently constructed list of $\mathcal{E}'$ and suppose that we constructed E'_j when we reached $E_{\delta_j} \in \mathcal{E}$. If E_{i+1} is obtained from E_i by swapping lines across some border b_k, i.e. by swapping the b_kth line α with either the $(b_k - 1)$th line β or the $(b_k + 1)$th line γ, then we create E'_{j+1} from E'_j by swapping α with β or γ, as appropriate; we also set $\delta_{j+1} = i + 1$. It can be shown that E'_j approximates any list $E_k \in \{E_{\delta_j}, E_{\delta_j+1}, \ldots, E_{\delta_{j+1}-1}\}$ in the following sense: for any two successive borders b_l and b_{l+1}, the (unordered) set of lines at positions $b_l + 1, \ldots, b_{l+1}$ in E'_j is the same as the (unordered) set of lines in E_k at these same positions. Moreover, it can be shown that the border-lines in E'_j and E_k are the same; this implies that the border-lines in $\{E_{\delta_j}, E_{\delta_j+1}, \ldots, E_{\delta_{j+1}-1}\}$ are the same and can be totally ordered. Finally, using k-set theory, it can be proved that $|\mathcal{E}'| = O(n^{2.5-\mu})$.

The data structure consists of (1) a red-black tree T'_j storing the total order of the border lines of E'_j, (2) an instance D'_j of the generalized 1-dimensional range searching structure built on the colored integers $I_j : (1, 2, \ldots, n)$, where integer p gets the color of the pth line of E'_j, and (3) a red-black tree I'_j storing I_j. We make all these structures persistent.

Given q, suppose we need to query E'_j. We find the smallest border b_s on or above q's upper endpoint and, symmetrically the greatest border b_g. We query the structure D'_j with $[b_s + 1, b_g]$. Then we scan all the integers $b_s, b_s - 1, \ldots, b_{s-1} + 1$ in I_j and report the distinct colors of the lines of E'_j at these positions that are intersected by q; symmetrically for b_g.

Theorem 3.2 *A set S of n colored lines in $\mathcal{R}^2$ can be stored in a data structure of size $O(n^{2.5-\mu} \log n)$ such that the i distinct colors of the lines that are intersected by a vertical query line segment can be reported in $O(n^\mu + i)$ time. Here μ is a tunable parameter in the range $0.5 < \mu < 1$. The problem is also solvable in $O(n^2 \log n)$ space and $O(\log n + i)$ query time.* □

We remark that our method is based on an idea used in [AvKO93] for a different (standard) problem. However, there are several crucial differences: (i) because our problem is a generalized one, our algorithm for constructing $\mathcal{E}'$ uses a different swap criterion, (ii) because our query is a vertical line segment rather than a ray (as in [AvKO93]), our choice of borders needs to be different in order to get a sublinear query time. Further discussion of this appears in [GJS93c].

3.3 Querying colored line segments with a line segment

3.3.1 Vertical query line segments

We can use an approach similar to the one at the beginning of Section 3.2. However, since we are now dealing with line segments rather than lines, we must overcome a subtle problem that can arise. We discuss this later. We draw vertical lines through the endpoints and the intersection points of the segments of S. Within any strip, the segments that cross it can be totally ordered. We sweep over the strips starting at the leftmost nonempty strip. Let $s_1, s_2, \ldots, s_m$ be the segments that cross this strip, sorted from bottom to top. For $1 \leq i \leq m$, we give s_i a label $l(s_i) = i$ and give this label the color of s_i. We store the segments $s_1, \ldots, s_m$ in this order in a partially persistent red-black tree T_S. We also store the colored labels $l(s_i)$, $1 \leq i \leq m$, in a partially persistent version T_l of the data structure of [GJS93a] for the generalized 1-dimensional range reporting problem.

Suppose we sweep from the ith to the $(i+1)$th strip. The cases where we encounter a right endpoint of a segment or the intersection point of two segments are easy to handle [GJS93c]. If we encounter the left endpoint of segment s, then in the current version of T_S, we locate s. Let t and u be the segments that are immediately below and above s in the $(i+1)$th strip. We insert s into the current version of T_S and store with it a label $l(s)$ that lies between $l(t)$ and $l(u)$. Moreover, we give the label $l(s)$ the same color as s and insert this colored number into the current version of T_l.

In this scheme, we need to assign special labels to the segments as we encounter them because all the segments are not present in each strip. However, we must be careful in choosing the labels since otherwise we may end up getting labels consisting of $\Theta(n)$ bits. Towards this end, we use a labeling scheme due to Dietz and Sleator [DS87]. Using their approach we take integer labels in the range $[0..O(n^2)]$, i.e., labels consisting of only $O(\log n)$ bits. We need to give segment s a label that lies in between $l(t)$ and $l(u)$. Using the scheme of [DS87], this may result in the relabeling of other segments. Dietz and Sleator show how to choose the labels such that only $O(1)$ amortized relabelings are necessary per update. If we relabel segment s from $l(s)$ to $l'(s)$, then we just delete the colored number $l(s)$ from T_l and insert the number $l'(s)$, having the same color as $l(s)$, into it.

Now let q be a vertical query segment. We locate the strip containing q and then search in the version of T_S corresponding to this strip for the lowest and highest segments s and t that intersect q. Finally, we search in the version of T_l corresponding to this strip for the distinct colors of all labels that are contained in the interval $[l(s), l(t)]$.

Theorem 3.3 *A set S of n colored line segments* *in the plane can be preprocessed into a data structure of size $O((n+\chi)\log n)$ such that the i distinct colors of the segments intersected by a vertical query line segment q can be reported in $O(\log n + i)$ time. Here χ, $0 \leq \chi \leq \binom{n}{2}$, is the number of pairwise intersections among the segments in S.* □

3.3.2 Arbitrary query line segments

If q is not vertical, then we use a different approach which we sketch briefly. We break up the input segments at their χ intersection points and store them in a segment tree T. For any node $v \in T$, the set $S(v)$ of segments allocated to v can be totally ordered within $Strip(v)$ as $s_1, \ldots, s_m$, from bottom to top. Suppose $q = \overline{ab}$ is such that $a, b \in Strip(v)$, with a below b. Let s_u be the lowest segment of $S(v)$ that is above a; define s_w symmetrically w.r.t. b. Clearly it suffices to report the distinct colors in $\{s_u, s_{u+1}, \ldots, s_w\}$, which we can solve using a structure for generalized 1-dimensional range searching. If $q = [a, b]$ is not as above, then we can identify a set V of $O(\log n)$ nodes in T such that for each $v \in V$, the segment $q_v = q \cap Strip(v)$ is as above. We simply query at each such v with q_v.

Theorem 3.4 *A set S of n colored line segments in $\mathcal{R}^2$ can be preprocessed into a data structure of size $O((n+\chi)\log n)$ so that the i distinct colors of the segments that are intersected by a query segment q can be reported in $O(\log^2 n + i\log n)$ time. Here χ, $0 \leq \chi \leq \binom{n}{2}$ is the number of pairwise intersections between segments in S.* □

4 Generalized triangle stabbing

We consider the following problem: "Preprocess a set S of n colored triangles in $\mathcal{R}^2$, so that the i distinct colors of the triangles stabbed by any query point q can be reported efficiently."

Wlog assume that each triangle $t \in S$ has a horizontal side. We group the triangle vertices into $\Theta(n^{1/2})$ vertical strips each of size $\Theta(n^{1/2})$. The triangles that intersect any strip V_i form two disjoint subsets T_i and T_i', where T_i (resp. T_i') consists of triangles having no (resp. at least one) vertex inside V_i. We further subdivide V_i into vertical strips by taking each triangle in T_i' and drawing a vertical line through each of its vertices that lies in V_i. Let W_{ij} be any such substrip within V_i and let T_{ij} consist of the triangles of T_i' that cross W_{ij}. Note that, by construction, no triangle of T_{ij} can have a vertex inside W_{ij}. (This partitioning technique is reminiscent of a method used in [OY88] for computing the measure of the union of iso-oriented boxes in $\mathcal{R}^d$.)

Given the query point q, suppose that $q \in V_i$ and $q \in W_{ij}$. Then we need to only report the distinct colors of the triangles of T_i and of T_{ij} that are stabbed by q. We discuss how to do this for V_i and T_i. (The discussion for W_{ij} and T_{ij} is similar.)

For any triangle $t \in T_i$, let $p(t)$ be the vertex of t shared by the horizontal side $h(t)$ of t and the slanted side $s(t)$ of t which crosses V_i. Let $w(t)$ be the wedge defined by $p(t)$ and (the extensions of) $h(t)$ and $s(t)$. Consider the set $R_i \subseteq T_i$ that yields right-facing wedges. Clearly, q stabs $t \in R_i$ iff the horizontal, leftward directed ray from q intersects $w(t)$. We can now solve the resulting generalized ray-rightwedge intersection problem by computing left envelopes and using a segment tree (as in Section 3.1). Symmetrically for left-facing wedges.

Theorem 4.1 *A set S of n colored triangles in $\mathcal{R}^2$ can be stored in a data structure of size $O(n^{3/2}\log n)$ such that the i distinct colors of the triangles that are stabbed by a query point q can be reported in $O(\log^2 n + i)$ time.* □

5 Generalized triangle range searching

We wish to preprocess a set S of n colored points in $\mathcal{R}^2$ so that the i distinct colors of the points contained in a query triangle q can be reported efficiently. For arbitrary q we note that the problem can be solved in $O(n^{2+\epsilon})$ space and $O(\log n + i)$ query time using constant-depth cutting trees. In the rest of this section, we give an efficient solution for query triangles that are fat. We define a *fat-triangle* to be a triangle in which each internal angle is at least γ for some constant $\gamma > 0$.

We begin with a solution for a query *fat-wedge* q. i.e., a wedge where the internal angle at the vertex v_q of q is at least γ. For now assume that q is y-monotone. We store the points of S at the leaves of a balanced binary search tree T by non-decreasing

y-coordinates from left to right. We augment each node v of T with an instance $HP(v)$ of the structure of Theorem 2.1 for generalized halfplane range reporting; $HP(v)$ is built on the points in v's descendant leaves.

Given q, we divide it into an upper wedge q_a and a lower wedge q_b, by drawing a horizontal line L through v_q. Consider q_a. Let l_a be the line containing the slanted side of q_a. We search in T using the y-coordinate of v_q and determine a set V_a of nodes that lie to the right of the search path. We query $HP(v)$ at each $v \in V_a$ with the halfplane l_a^-. Symmetrically for q_b. It can be shown that the space used is $O(n \log^2 n)$ and the query time is $O(\log^3 n + i \log n)$.

What if q is not y-monotone? In preprocessing, we select $t = \lceil 2\pi/\gamma \rceil$ coordinate systems $\mathcal{C}_i = (x_i y_i)$, where all the $\mathcal{C}_i$ share the same origin and $\mathcal{C}_{i+1}$ is offset from $\mathcal{C}_i$ by an angle γ, $0 \leq i \leq t-1$ (indices are modulo t.) Within each $\mathcal{C}_i$ we build an instance of the data structure for y_i-monotone fat-wedges. Given a query fat-wedge q, we locate a $\mathcal{C}_i$ such that q is y_i-monotone and then query the associated structure. (It is easily seen that such a $\mathcal{C}_i$ exists.)

Now consider the case of a query fat-triangle q. We store the points by non-decreasing x-coordinates from left to right in a balanced search tree T' and augment each node v with an instance $FW(v)$ of the structure given above for fat-wedges, which is built on the points in v's descendant leaves. Given q, we divide it into two triangles q_l and q_r, each with a vertical side s, with q_l to the left of s and q_r to the right. We search in T' with the x-coordinate of s and identify a set V_l of nodes that lie to the left of the search path. For each node $v \in V_l$, we query $FW(v)$ with the wedge supporting q_l, which is a fat-wedge. Symmetrically for q_r.

Theorem 5.1 *Let $\gamma > 0$ be a constant. A set S of n colored points in $\mathcal{R}^2$ can be stored in a data structure of size $O(n \log^3 n)$ such that the i distinct colors of the points that are contained in a query triangle q each of whose internal angles is at least γ can be reported in time $O(\log^4 n + i \log^2 n)$.* □

6 Conclusions and further work

We have presented efficient solutions to several generalized intersection problems involving non-iso-oriented objects. Our methods have included sparse representations, persistence, and filtering search.

Besides improving our bounds, three problems are of particular interest: (i) obtaining dynamic data structures for the generalized problems considered here, (ii) obtaining linear-space or near linear-space solutions with output-sensitive query times (of the form $O(n^p + i)$ or $O(n^p + i \cdot polylog(n))$, $0 < p < 1$) for the generalized halfspace range searching problem in $d \geq 4$ dimensions and for the generalized simplex range searching problem in $d \geq 2$ dimensions, and (iii) solving the counting versions of Problems 3–7 in Table 1.

References

[AHL90] A. Aggarwal, M. Hansen, and T. Leighton. Solving query-retrieval problems by compacting Voronoi diagrams. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 331–340, 1990.

[AvK93] P.K. Agarwal and M. van Kreveld. Connected component and simple polygon intersection searching. In *Proceedings of the 1993 Workshop on Algorithms and Data Structures*, pages 36–47, August 1993.

[AvKO93] P.K. Agarwal, M. van Kreveld, and M. Overmars. Intersection queries for curved objects. *Journal of Algorithms*, 15:229–266, 1993.

[CGL85] B.M. Chazelle, L.J. Guibas, and D.T. Lee. The power of geometric duality. *BIT*, 25:76–90, 1985.

[DS87] P.F. Dietz and D.D. Sleator. Two algorithms for maintaining order in a list. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 365–372, 1987.

[DSST89] J.R. Driscoll, N. Sarnak, D.D. Sleator, and R.E. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38:86–124, 1989.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer–Verlag, 1987.

[GJS93a] P. Gupta, R. Janardan, and M. Smid. Further results on generalized intersection searching problems: counting, reporting, and dynamization. In *Proceedings of the 1993 Workshop on Algorithms and Data Structures*, pages 361–372, August 1993.

[GJS93b] P. Gupta, R. Janardan, and M. Smid. On intersection searching problems involving curved objects. Technical Report TR–93–42, Dept. of Computer Science, University of Minnesota, 1993. Submitted.

[GJS93c] P. Gupta, R. Janardan, and M. Smid. Efficient algorithms for generalized intersection searching with non-iso-oriented objects. Technical Report TR–93–73, Dept. of Computer Science, University of Minnesota, 1993. Submitted.

[JL93] R. Janardan and M. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry & Applications*, 3:39–69, 1993.

[Mat91a] J. Matoušek. Cutting hyperplane arrangements. *Discrete & Computational Geometry*, 6:385–406, 1991.

[Mat92a] J. Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992.

[Mat92b] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proceedings of the 8th Annual ACM Symposium on Computational Geometry*, pages 276–285, 1992.

[Mul93] K. Mulmuley. *Computational Geometry: An introduction through randomized algorithms.* Prentice–Hall, 1993.

[OY88] M.H. Overmars and C.K. Yap. New upper bounds in Klee's measure problem. In *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, pages 550–556, 1988.

Video Review

3rd Annual Video Review of Computational Geometry

Computational geometry concepts are often easiest to understand visually. Indeed, most papers in computational geometry rely on diagrams to communicate the intuition behind their results. However, static figures are not always adequate to describe geometric algorithms, since algorithms are inherently dynamic. The accompanying videotape showcases advances in the use of algorithm animation, visualization, and interactive computing in the study of computational geometry. The following pages contain short descriptions of the eight segments in the videotape.

The eight video segments present a variety of geometric algorithms and concepts, organized by dimensionality. The first three segments are animations of two-dimensional algorithms: a polygonal approximation algorithm from cartography, an algorithm for computing the rectangle discrepancy, and a fixed-radius neighbor-searching algorithm. The fourth video presents the GASP system for animating geometric algorithms in two or three dimensions; GASP is used by two other segments in the review. The last four segments illustrate algorithms and concepts in three or more dimensions: The fifth segment explores the relationship between penumbral shadows in three dimensions and four-dimensional polytopes. The final three segments show three-dimensional algorithms for interactive collision detection, polyhedral separators, and interactive display of alpha hulls.

We thank the members of the Video Program Committee for their help in evaluating the entries. The members of the committee were Marshall Bern (Xerox PARC), Marc Brown (DEC SRC), Leo Guibas (Stanford), John Hershberger (Mentor Graphics), Jim Ruppert (NASA Ames), and Jenny Zhao (Silicon Graphics).

We also thank Bob Taylor of the DEC Systems Research Center for his support in publishing the 1st and 2nd Annual Video Reviews. Both are available free of charge upon request. The '92 Review is SRC Research Report #87a (a written guide) and #87b (the videotape), and the '93 Review is reports #101a and #101b. You can obtain either by sending electronic mail to `src-report@src.dec.com`, or physical mail to Librarian, DEC Systems Research Center, 130 Lytton Ave., Palo Alto, CA 94301. You can also get the written guides by anonymous ftp from `gatekeeper.dec.com` in the directory `pub/DEC/SRC/research-reports`.

Marc H. Brown
John Hershberger
Video Review Co-Chairs

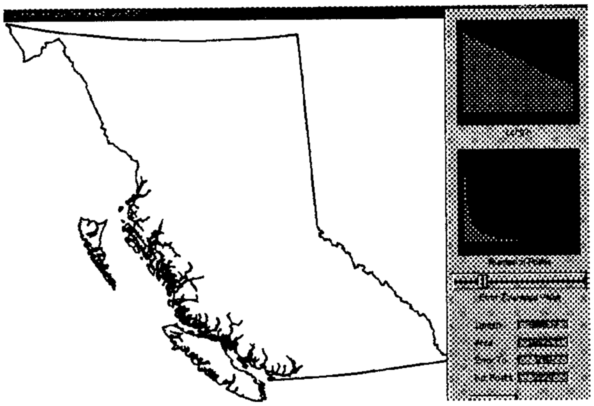
An $O(n \log n)$ Implementation Of The Douglas-Peucker Algorithm For Line Simplification

John Hershberger
Mentor Graphics

Jack Snoeyink
Department of Computer Science
University of British Columbia

Introduction

An important task of the cartographer's art is to extract features from detailed data and represent them on a simple and readable map. As computers become increasingly involved in automated cartography, efficient algorithms are needed for the tasks of extraction and simplification. Cartographers [1, 6] have identified the ***line simplification problem*** as an important part of representing linear features. Given a ***polygonal chain***, a sequence of vertices $V = \{v_0, v_1, \ldots, v_n\}$, the problem is to find a subsequence of vertices that approximates the chain V. We assume that the input chain V has no self-intersections (but not the output).



In 1973, Douglas and Peucker (now Poiker) [3] published a simple recursive simplification method in the cartography literature that has become the favorite of many [5, 10]. This method has been independently proposed in other contexts such as vision [8] and computational geometry [9]. Our video animates two different implementations of this simplification method—the straightforward implementation suggested in Douglas and Peucker and the implementation of Hershberger and Snoeyink that improves the worst-case running time.

We describe the method, the implementations, and the animation.

Method

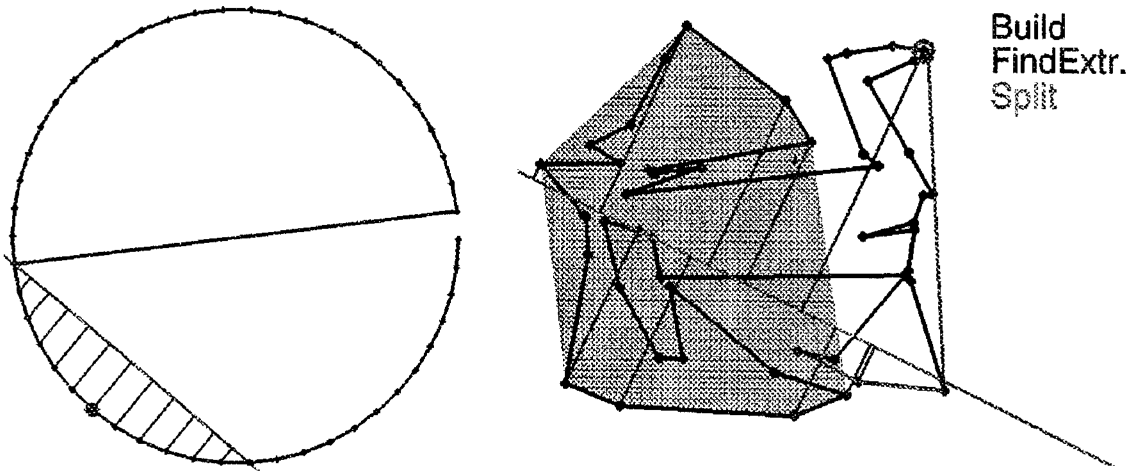
Given a polygonal chain $V = \{v_0, v_1, \ldots, v_n\}$, the recursive line simplification method proceeds as follows: Initially, approximate V by the line segment $\overline{v_0v_n}$. Determine the farthest vertex v_f from the line $\overleftrightarrow{v_0v_n}$. If its distance $d(v_f, \overleftrightarrow{v_0v_n})$ is at most a given tolerance $\varepsilon \geq 0$, then accept the segment $\overline{v_0v_n}$ as a good approximation to V. Otherwise, break V at v_f and recursively approximate the subchain $\{v_0, v_1, \ldots, v_f\}$ and $\{v_f, \ldots, v_n\}$.

Algorithms

The two implementations differ in the way they find v_f, the farthest vertex from $\overleftrightarrow{v_0v_n}$.

The straightforward approach measures distance from $\overleftrightarrow{v_0v_n}$ for each vertex and takes the maximum. Thus, the running time of this algorithm will satisfy a quicksort-like recurrence, with best case $\Theta(n \log n)$. Since geometry determines the farthest vertex, the worst-case running time is $\Theta(n^2)$. One cannot use linear-time median finding or randomization to improve the running time. (If one assumes that the input data is such that farthest vertices are found near the middle of chains, then one expects $O(n \log n)$ behavior.)

The second implementation uses a ***path hull data structure*** [2, 4] to maintain a dynamic convex hull of the polygonal chain V. Using Melkman's convex hull algorithm [7], we compute two convex hulls from the middle of the chain outward. We can find



a farthest vertex v_f from a line by locating two extreme points on each hull using binary search. When we split V at vertex v_f, we undo one of the hull computations to obtain the hull from the "middle" to v_f, recursively approximate the subchain containing the middle, then build hulls for the other subchain and recursively approximate it. One can use a simple credit argument to show that the total time taken by the algorithm is $O(n \log n)$. The best case can even be linear, if all the hulls are small and splits occur at the ends.

Unfortunately for us, the statistical properties of cartographic data usually means that the straightforward implementation is slightly faster than the path hull implementation. Since the variance of the running time of the path hull implementation is smaller, it may be interesting for parallel or interactive applications.

Animation

The programs were run on a Silicon Graphics Crimson in the GraFiC lab at UBC and output was recorded to video in real time.

Acknowledgements

Scott Andrews extracted the data of the coastline of British Columbia from the Digital Chart of the World. NSERC and the B.C. Advanced Systems Institute supported the work on this video. DEC Systems Research Center has also provided partial support for the research.

References

[1] B. Buttenfield. Treatment of the cartographic line. *Cartographica*, 22:1–26, 1985.

[2] D. Dobkin, L. Guibas, J. Hershberger, and J. Snoeyink. An efficient algorithm for finding the CSG representation of a simple polygon. *Algorithmica*, 10:1–23, 1993.

[3] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.

[4] J. Hershberger and J. Snoeyink. Speeding up the Douglas-Peucker line simplification algorithm. In *Proc. 5th Intl. Symp. Spatial Data Handling*, pages 134–143. IGU Commission on GIS, 1992.

[5] R. B. McMaster. A statistical analysis of mathematical measures for linear simplification. *Amer. Cartog.*, 13:103–116, 1986.

[6] R. B. McMaster. Automated line generalization. *Cartographica*, 24(2):74–111, 1987.

[7] A. A. Melkman. On-line construction of the convex hull of a simple polyline. *Info. Proc. Let.*, 25:11–12, 1987.

[8] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Comp. Vis. Graph. Image Proc.*, 1:244–256, 1972.

[9] G. Rote. Quadratic convergence of the sandwich algorithm for approximating convex functions and convex figures in the plane. In *Proc. 2nd Can. Conf. Comp. Geom.*, pages 120–124, Ottawa, Ontario, 1990.

[10] E. R. White. Assessment of line-generalization algorithms using characteristic points. *Amer. Cartog.*, 12(1):17–27, 1985.

Computing the Rectangle Discrepancy *

David P. Dobkin Dimitrios Gunopulos
Department of Computer Science
Princeton University
Princeton, NJ 08544

This is the animation of an algorithm that computes the rectangle discrepancy of a two-dimensional point set. Our main motivation comes from sampling problems in computer graphics. Supersampling is one of the most general ways of attacking antialiasing. In this approach we sample the picture at a very high rate, and then we resample by averaging supersamples within the pixel area to compute the pixel value. Ray tracing, a technique used to produce realistic images of computer modeled scenes, is another instance of supersampling. Here we sample the radiance at a set of points in each pixel.

The discrepancy theory provides a measure of the quality of the sampling patterns. The rectangle discrepancy in particular gives a good measure on how well a sample pattern, when applied to the area of one or more pixels, captures small details in the picture. This algorithm allows us to find sampling sets with very low rectangle discrepancy. Interestingly the problem of computing the rectangle discrepancy also arises in the context of learning theory [4].

The algorithm is presented in [2]. We give the problem and we show the main techniques used in the algorithm. Finally we run the algorithm on a sample point set.

Let us consider a point set $\mathcal{X}$ in the unit square. Let $\mathcal{F}$ be the set of all rectangles in the unit square with edges parallel to the two axis. For an axis oriented rectangle A ($A \in \mathcal{F}$), we define the function

$$In(A) = |A \cap \mathcal{X}|$$

The *rectangle discrepancy* of a rectangle $A \in \mathcal{F}$ is

$$\mathcal{D}(A) = |In(A)/|\mathcal{X}| - Area(A)|$$

The *maximum rectangle discrepancy* of the set $\mathcal{X}$ is

$$Max\mathcal{D}(\mathcal{X}) = max_{A \in \mathcal{F}}(\mathcal{D}(A))$$

We also define the following simpler functions for $A \in \mathcal{F}$

$$\mathcal{D}_l(A) = In(A)/|\mathcal{X}| - Area(A)$$

and

$$\mathcal{D}_o(A) = Area(A) - In(A)/|\mathcal{X}|$$

The algorithm we present in this animation computes

$$Max\mathcal{D}_l(\mathcal{X}) = max_{A \in \mathcal{F}}(\mathcal{D}_l(A))$$

for a given input set $\mathcal{X}$, in time $O(|\mathcal{X}|^2 \log |\mathcal{X}|)$. The same techniques however can be used to maximize $\mathcal{D}_o$, and so this problem is equivalent to finding $Max\mathcal{D}(\mathcal{X})$.

We explain the main idea of the algorithm in a series of simple lemmata. First we show that we have to consider at most $O(|\mathcal{X}|^4)$ rectangles, more specifically the ones that have all their edges pass through a point in $\mathcal{X}$.

This allows us to study pairs of points in in $\mathcal{X}$. For each such pair we find the rectangle that maximizes $\mathcal{D}_l$ over all rectangles with horizontal edges that pass through the points of the given pair.

This problem is much easier now because we know the y coordinates of the rectangle. Assuming that the two points are p_b, p_t ($p_b(y) < p_t(y)$), and the height of the rectangle is dy, ($dy = p_t(y) - p_b(y)$) we have to find the following maximum

$$max_{A \in \mathcal{F}_{ij}}(dx_A\, dy - In(A)/|\mathcal{X}|)$$

where $\mathcal{F}_{ij}$ is the set of rectangles that pass through points p_b and p_t, and dx_A is the width of A. If we eliminate the points in $\mathcal{X}$ that either lie above the top y coordinate of the rectangle or below the lower y coordinate, this becomes an one-dimensional problem. We project the points on the x axis and obtain a new set $\mathcal{X}'$ of one-dimensional points. We can equivalently maximize the following function

$$max_{A \in [0,1], p_b(x) \in A}(Length(A) - In(A)/|\mathcal{X}'|)$$

We can reduce this problem to the problem of computing the halfspace discrepancy of one-dimensional point sets, a simpler problem studied in [1]. Here we want to find the

*This work supported in part by the National Science Foundation under Grant Number CCR93-01254 and by The Geometry Center, University of Minnesota, an STC funded by NSF, DOE, and Minnesota Technology, Inc.

anchored interval that maximizes

$$max_{A=[0,z_i]}(z_i - i/|\mathcal{Z}|)$$

for a given one-dimensional set $\mathcal{Z} = \{z_1 \ldots z_n\}$ with $0 \leq z_i < z_{i+1} \leq 1$. This is a linear function of the rank and the coordinate of the point. To solve it we find the convex hull of the set $\{(z_1, 1) \ldots (z_n, n)\}$ and find the maximum with a binary search. From this maximum we obtain the solution to the original maximization problem.

In this way we compute the maximum rectangle for all pairs of points. Clearly the rectangle that maximizes $\mathcal{D}_l$ is the maximum of these $O(|\mathcal{X}|^2)$ rectangles.

Finally, we show how to use the dynamic data structure of [5] to solve a sequence of one-dimensional problems efficiently. With the use of this technique we can maintain the convex hull at a cost of $O(\log |\mathcal{X}|)$ time per insertion. This in turn allows us to find the maximum rectangle for each pair of points in $O(\log |\mathcal{X}|)$ time, and yields the final running time of $O(|\mathcal{X}|^2 \log |\mathcal{X}|)$.

This video was prepared in the Computer Science department at Princeton University. The implementation of the algorithm was written in C, to run under UNIX on a Silicon Graphics Iris workstation. The animation was created with the animation system GASP that Ayellet Tal and David Dobkin are developing in this department [3]. Figure 1 shows the animation running on a sample input set. The rectangle that maximizes $\mathcal{D}_l$ is now displayed on the screen. Recording was done at the Interactive Computer Graphics Lab at Princeton and editing was done with the assistance of the Department of Media Services at Princeton.

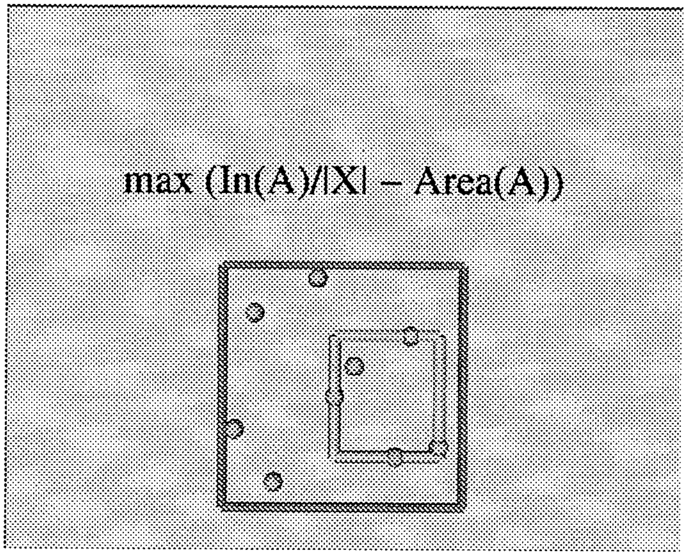


Fig. 1: The rectangle that maximizes $\mathcal{D}_l$.

REFERENCES

[1] D.P. Dobkin and D. Eppstein, Computing the Discrepancy. *Proceedings of the ninth annual Symposium on Computational Geometry,* (1993).

[2] D.P. Dobkin and D. Gunopulos, Computing the Rectangle Discrepancy. *Princeton University Technical Report 443-94.*

[3] D.P. Dobkin and A. Tal, GASP – A System to Facilitate Animating Geometric Algorithms. *Third Annual Video Review of Computational Geometry, to appear,* (1994)

[4] W. Maass, Efficient Agnostic PAC-Learning with Simple Hypotheses. *Submitted to COLT '94.*

[5] Preparata, F. An optimal real time algorithm for planar convex hulls. *Comm. ACM 22,* (1979).

An animation of a fixed-radius all-nearest-neighbors algorithm*

Hans-Peter Lenhof Michiel Smid

This video shows an animation of an algorithm due to the authors [2] for solving the *fixed-radius all-nearest-neighbors problem.* In this problem, we are given a set S of n points and a real number δ, and we have to report all pairs of points that are at distance at most δ. The algorithm works in two stages. In the first stage, a grid is computed. Instead of a standard grid, we use a so-called degraded grid that is easier to construct by means of a simple sweep algorithm. This degraded grid consists of boxes with sides of length at least δ. If a box contains points of S, then its sides are of length exactly δ. In the second stage, this grid is used for the actual enumeration. For each non-empty box, it suffices to compare each of its points with all points in the same box or in one of the neighboring boxes. Although the algorithm may compare many pairs having distance more than δ, it can be shown that the total number of pairs considered is proportional to the number of pairs that are at most δ apart. As a result, the total running time of the algorithm is proportional to $n \log n$ plus the number of pairs that are at distance at most δ.

As an application, we give an animation of the algorithm of Heiden at al.[1] for triangulating the contact surface of a molecule. In a first step, points on this surface are computed. Given these points, we compute all pairs that are at distance at most δ for a suitable choice of δ. This gives for each point a neighbor list containing all points that are at distance at most δ to it. Given these lists, we can triangulate the surface incrementally: An edge of the current triangulation is called an outer edge, if it belongs to only one triangle. All outer edges are maintained in a queue. With each edge, we store a so-called common neighbor list, which is the intersection of the two neighbor lists of the points belonging to this edge. We start with any triangle. Its edges and their common neighbor lists are inserted into the queue. As long as the queue is non-empty, we do the following: We take any edge from the queue and look at all points in its common neighbor list. Each such point defines a triangle with the current edge. We take that point defining the smallest triangle. This triangle becomes part of the triangulation and we update the queue appropriately.

The video was produced at the Max-Planck-Institute for Computer Science, using Silicon Graphics Indy and Indigo^2 machines, and using Showcase and Geomview-1.1 software. We thank Roland Berberich, Jack Hazboun, Dirk Louis and Peter Müller for their help in producing this video.

References

[1] W. Heiden, M. Schlenkrich and J. Brickmann. *Triangulation algorithms for the representation of molecular surface properties.* J. Comp. Aided Mol. Des. 4 (1990), pp. 255-269.

[2] H.P. Lenhof and M. Smid. *Enumerating the k closest pairs optimally.* Proceedings FOCS 1992, pp. 380-386.

*This work was supported by the ESPRIT Basic Research Actions Program, under contract No. 7141 (project ALCOM II). Authors' address: Max-Planck-Institut für Informatik, D-66123 Saarbrücken, Germany.

GASP - A System to Facilitate Animating Geometric Algorithms [*]

Ayellet Tal David Dobkin
Department of Computer Science
Princeton University
Princeton, NJ 08544

The GASP system helps in the creation and viewing of animations for geometric algorithms. The user need not have any knowledge of computer graphics in order to quickly generate an animation. GASP allows the fast prototyping of algorithm animations. A typical animation can be produced in a matter of days or even hours. Even highly complex geometric algorithms can be animated with ease. The system is also intended to facilitate the task of implementing and debugging geometric algorithms.

The application's code specifies only the structure of the animation (which building blocks are included), and the structure of each scene (e.g., the position of the objects). The user need not be concerned with the way the animation appears on the screen. If the user wishes to influence the look of the animation and not only its structure, he can edit an ASCII *"Style File"* which controls viewing aspects of the animation. Even when the user changes the style file, the user needs no specific knowledge of computer graphics. The animation is generated automatically by the system. But a different animation will be generated if the style file is modified.

The GASP's environment allows the viewer to control the execution of the algorithm in an easy way. The animation can be run at varying speeds: fast(>>), slow(>) or step by step (> |). The analogous <, << and | < push buttons run the algorithm in reverse. The viewer can *PAUSE* at any time to suspend the execution of the algorithm or can *EJECT* the movie. The environment is invaluable for education and debugging.

This video shows five examples of GASP in action. These examples were chosen to give a sense of the range of possible applications of GASP. In addition, two other videos [1], [3] that use our system appear in this collection of animations.

To begin, we show a clip from [6]. This is the hierarchical data structure for representing convex polyhedra [4, 5]. We explain a single pluck and then show how the hierarchy progresses from level to level. To create the animation for explaining a single pluck the user need only write the following short code. The animation was an aid in debugging the algorithm for detecting plane-polyhedral intersection.

```
explain_pluck(poly_vert_no, poly_vertices,
       poly_face_no, poly_faces, poly_names,
       vert_no, vertices, face_no, faces)
{
  /* create the polyhedron */
  Begin_atomic("poly");
  Create_polyhedron("PQ", poly_vert_no,
     poly_face_no, poly_vertices, poly_faces);
  Rotate_world();
  End_atomic();

  /* remove vertices and cones */
  Begin_atomic("pluck");
  Split_polyhedron(poly_names, ``PQ'',
                 vert_no, vertices);
  End_atomic();

  /* add new faces */
  Begin_atomic("add_faces");
  Add_faces(poly_names[0], face_no, faces);
  End_atomic();

  /* undo plucking */
  Undo(2);
}
```

The second animation, based on [10] is a remake of [9]. The animation shows a configuration of six tetrahedra that cannot be taken apart by translation with two hands. Then, it presents a configuration of thirty objects that cannot be taken apart by applying an isometry to any proper subset. The purpose of the animation is to illustrate the use of GASP as an illustration tool for geometric configurations. It took us far less than a day to generate that animation. The animation was created by the following function.

[*] This work supported in part by the National Science Foundation under Grant Number CCR93-01254 and by The Geometry Center, University of Minnesota, an STC funded by NSF, DOE, and Minnesota Technology, Inc.

```
hands(){
  for (i = 0; i ≤ stick_no; i++){
    /* stick i */
    get_polyhedron(&points, &indices, &nmax,
          &fmax, &atomic_name, &stick_name);

    Begin_atomic(atomic_name);
    Create_polyhedron(stick_name, nmax, fmax,
          points, indices);
    End_atomic();
  }

  Begin_atomic("Rotate");
  Rotate_world();
  End_atomic();
}
```

The following is part of the style file for the above animation. The style file determines the following aspects of the animation. The background color is light gray. The colors to be chosen by GASP are colors which fit the creation of a video (rather than the screen). Each atomic unit spans 10 frames. Rotation of the world is done 380 degrees around the Y axis, and it is executed over 160 frames, instead of over 10. The color of the stick which fades in during atomic unit stick0 is red. (Similarly, the style file includes colors for the other sticks.)

```
begin_global_style
  background = 0.9 0.9 0.9;
  color = VIDEO;
  frames = 10;
end_global_style

begin_unit_style Rotate
  frames = 160;
  rotation_world = Y 380.0;
end_unit_style

begin_unit_style stick0
  color = 0.625 0.273475 0.273475;
end_unit_style
```

Next we show an animation of a motion planning algorithm [7]. It is a repeat of the first part of [8]. The object to be moved is a disc and the obstacles are simple disjoint polygons. The goal of the animation is to show the use of GASP in creating two-dimensional animations.

The forth animation, which is based on [2], shows a line segment intersection algorithm in action and illustrates its most important features. The animation consists of three phases. The first phase presents the initial line segments and the visibility map that needs to be built. The second phase demonstrates that the visibility map is being constructed by operating in a sweepline fashion, scanning the segments from left to right, and maintaining the vertical visibility map of the region swept along the way. The third phase demonstrates that the cross section along the sweepline is maintained in a lazy fashion. The animation is used as an aid in explaining a highly complex algorithm and to allow students to interact with the algorithm. A student can choose the initial line segments by editing an ASCII file and view how the animation is changing.

The last clip animates heapsort. The colored rods have lengths to be sorted. We first display the initial creation of the heap. Next, simple motions are used to control the repeated operations of removing the largest element, and reheaping. Heapsort is given as an example for using GASP to facilitate the animation of any algorithm that involves the display of three dimensional geometries.

This video was prepared in the computer science department at Princeton University. The programs were written in C to run under UNIX on a Silicon Graphics Iris. Recording was done at the Interactive Computer Graphics Lab at Princeton and editing was done with the assistance of the Princeton Department of Media Services.

We thank Kirk Alexander and Mike Mills for their help in producing the final video.

REFERENCES

[1] H. Bronnimann. Almost optimal set covers in finite VC-dimensions. In *The Third Annual Video Review of Computational Geometry*, 1994.

[2] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM*, 39(1):1–54, 1992.

[3] D. Dobkin and D. Gunopulos. Computing the rectangle descrepency. In *The Third Annual Video Review of Computational Geometry*, 1994.

[4] D. Dobkin and D. Kirkpatrick. Fast detection of polyhedral intersections. *Journal of Algorithms*, 6:381–392, 1985.

[5] D. Dobkin and D. Kirkpatrick. Determining the separation of preprocessed polyhedra - a unified approach. *ICALP*, pages 400–413, 1990.

[6] D. Dobkin and A. Tal. Building and using polyhedral hierarchies (video). In *The Ninth Annual ACM Symposium on Computational Geometry*, page 394, May 1993.

[7] H. Rohnert. Moving a disc between polygons. *Algorithmica*, 6:182–191, 1991.

[8] S. Schirra. Moving a disc between polygons (video). In *The Ninth Annual ACM Symposium on Computational Geometry*, pages 395–396, May 1993.

[9] J. Snoeyink. Objects that cannot be taken apart with two hands (video). In *The Ninth Annual ACM Symposium on Computational Geometry*, page 405, May 1993.

[10] J. Snoeyink and J. Stolfi. Objects that cannot be taken apart with two hands. In *The Ninth Annual ACM Symposium on Computational Geometry*, pages 247–256, May 1993.

Penumbral Shadows

Adrian Mariano[*] Linus Upson[†]

The shadows cast by polygonal occluding objects illuminated by polygonal light sources are complicated. Unlike shadows cast by point sources, they are divided into regions where the shadow intensity changes at different rates. The boundaries between these different regions form the *shadow diagram.*

The video tape depicts shadows cast by a square light source with a square occluding object. We have included two types of images: flat shadow scenes and perspective views. The flat shadow images were calculated using the exact formula for radiative energy transfer. This calculation depends only on the boundary of the visible light source. For polygons, it can be reduced to a simple summation.[1]

Finding the boundary of the visible portion of the illuminating polygon when it is occluded by a second polygon requires computing the intersection of two polygons. General purpose algorithms for computing intersections of arbitrary polygons in the plane are quite complicated, so we used the Sutherland–Hodgman algorithm [2] which cannot handle concave polygons, but which is much simpler than the fully general methods [3]. We created the three-dimensional perspective views from the flat images using the texture mapping capabilities of *Rayshade.*

The shadow diagrams can be obtained by projecting the edges or vertices of the light source through vertices or edges of the occluder onto the plane of the shadow. The video illustrates that when a square occluding object is illuminated by a square light source, the shadow diagram is a projection of a four-dimensional hypercube.

In general, the shadow diagrams for an arbitrary light source and occluder can be seen as projections of four-dimensional polytopes by considering the lines which intersect the plane of the shadow. Let S be the plane of the shadow and let P be a different plane parallel to S. If a line intersects S at the point (x_s, y_s) and it intersects P at the point (x_p, y_p) then the line can be mapped to a point (x_s, y_s, x_p, y_p) in in $\mathbf{R}^4$. Now let D be the set of all lines which intersect both the light source and the occluder. When D is mapped into $\mathbf{R}^4$, the result is a four-dimensional polytope. If this polytope is projected into $\mathbf{R}^2$ by discarding the last two coordinates of each point, the shadow diagram results. Because P can be changed, this construction describes a particular shadow diagram as the projections of any one of a family of polytopes.

References

[1] Hottel and Sarofim. *Radiative Transfer.* McGraw Hill, New York. 1967.

[2] Foley, van Dam, Feiner and Hughes. *Computer Graphics Principles and Practice.* Addison-Wesley, New York. 1990.

[3] Vatti, Bala. "A Generic Solution to Polygon Clipping." *Communications of the ACM*, July 1992.

[4] Teller, Seth. "Computing the Antipenumbra of an Area Light Source." SIGGRAPH '92 Conference Proceedings, July 1992.

[*] adrian@cam.cornell.edu, Center for Applied Math, 657 ETC, Cornell University, NY 14853–3801.

[†] Linus_Upson@NeXT.com, NeXT, 900 Chesapeake Drive, Redwood City, CA 94063.

Exact Collision Detection for Interactive Environments

Jonathan D. Cohen
Computer Science Department
University of North Carolina
Chapel Hill, NC 27599-3175

Ming C. Lin
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

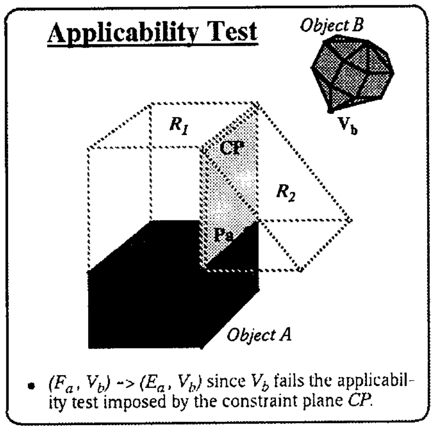
Dinesh Manocha *
Computer Science Department
University of North Carolina
Chapel Hill, NC 27599-3175

Madhav K. Ponamgi
Computer Science Department
University of North Carolina
Chapel Hill, NC 27599-3175

Abstract:
We demonstrate algorithms for collision detection in large-scaled, interactive environments. Such environments are characterized by the number of objects undergoing rigid motion and the complexity of the models. We do not assume that the motions of the objects are expressible as closed form functions of time, nor do we assume any upper bounds on their velocities.

Over the last few years, a great deal of interest has been generated in *virtual or synthetic* environments, such as architectural walkthroughs, visual simulation systems for designing the shapes of mechanical parts, training systems, etc. Interactive, large-scaled virtual environments pose new challenges to the collision detection problem, because they require performance at interactive rates for thousands, of pairwise intersection tests [1].

Our algorithms use a two-level hierarchical detection test for each model to *selectively* compute the *precise* contact between objects, achieving real-time performance without compromising accuracy. At the top level, we use a *dimension reduction* approach (based on a technique termed as *sweep and prune*) and the *temporal and geometric coherence* that exists between successive frames to overcome the bottleneck of $O(n^2)$ pairwise comparison tests in a large environment of n moving objects [1]. After we eliminate the objects pairs which are not in the close vicinity of each other, we employ an exact collision detection scheme. The exact convex polytope collision detection algorithm is based upon [2]: locality (captured by walking from one Voronoi region of the candidate feature to one of its neighboring feature's Voronoi region according to the constraints failed) and coherence (used to reduce computation efforts in a dynamic environment). We can deal with non-convex objects or articulated bodies by using a sub-part hierarchical tree representation.



Applicability Test

- (F_a, V_b) -> (E_a, V_b) since V_b fails the applicability test imposed by the constraint plane *CP*.

Figure 1: An Example of collision detection for convex polyhedra

• Exact Collision Detection

The Voronoi regions form a partition of space outside the polytope according to the closest feature. The collection of Voronoi regions of each polytope is the generalized Voronoi diagram of the polytope. Note that the generalized Voronoi diagram of a convex polytope has linear size and consists of polyhedral regions. Each Voronoi region is bounded by a set of constraint planes with pointers to the neighboring cells (which each share a constraint plane with it).

If a point lies on a constraint plane, then it is equidistant from the two features which share this constraint plane in their Voronoi regions. If a point P on object A lies inside the Voronoi region of the feature f_B on object B, then f_B is a closest feature to the point P.

Our method is straightforward. We start with a candidate pair of features, one from each polytope, and check whether the closest points lie on these features. Since the polytopes and their faces are convex, this is a local test involving only the neighboring features of the current candidate features. If either feature fails the test, we step to a neighboring feature of one or both candidates, and try

*Supported in part by a Junior Faculty Award and ARPA Contract #DAEA 18-90-C-0044

again. With some simple preprocessing, we can guarantee that every feature has a constant number of neighboring features. This is how we can verify or update the closest feature pair in *expected constant* time. Fig. 1 shows how this algorithm works on a pair of simple convex polytopes.

- **Sweep and Prune**

The exact polytope collision detection algorithm is fast enough to perform collision detection at real-time, in a multi-body environment of moderate size. However, the $O(n^2)$ pairwise detection tests become a bottleneck for large n. By exploiting the spatial and temporal coherence, we use a method whose running time is a linear function of the number of objects and the number of pairs of objects in close vicinity of each other.

For each object in the environment, we compute a smallest fitting, axis-aligned bounding box, which is dynamically resized using a hill-climbing algorithm based on the properties of convex polytopes and the principle of coherence.

These bounding boxes are used to prune down the number of pairwise collision detection tests. We take the orthogonal projections of the bounding boxes onto the x, y, and z axes and sort the intervals in three separate lists. If the projections overlap in *all three* dimensions, we activate the exact collision detection algorithm. At each frame, we sweep over the sorted lists of intervals, updating them and culling out the pairs of non-overlapping bounding boxes.

Each polytope in the environment now has a bounding box. We only call the exact collision detection algorithm between two objects when their bounding boxes overlap. Due of coherence, the sorted lists of intervals, constructed from the projections of the bounding boxes, change little from frame to frame, so we use a bubble or insertion sort to update these lists in expected linear time. Without the bounding box hierarchy, we would have to perform a quadratic number of pairwise exact detection tests.

- **Video Description**

We have successfully demonstrated the algorithm in an architectural walkthrough environment and a multi-body simulation system. For simulations consisting of thousands of models, the overall algorithm has exhibited linear time performance in practice.

We first demonstrate how the polytope collision detection algorithm [2] tracks the closest features between two convex polytopes in real time. We maintain and update the closest features using the Voronoi regions of the polytopes. Since the polytopes move only slightly between the frames, the temporal and geometric coherence is well preserved. Thus, the expected running time of this algorithm is constant, independent of the complexity of the polytopes.

This is followed by demonstration the algorithm on an articulated body, namely a hand, represented by the union of multiple convex polytopes. To demonstrate the efficiency of the algorithm and effectiveness of the coherence, we keep track of all pairs of closest features between each finger and the nearby objects. However, using a subpart hierarchical tree representation as in [2] can eliminate the unnecessary computation.

The over all collision detection algorithm is tested on a walkthrough and a multi-body dynamic simulation environment. The video footage of walkthrough was taken from a user's **actual walkthrough experience**. The collision detection algorithm performs very well in large environments composed of hundreds of polytopes. The kitchen, like most realistic environments, is fairly static; therefore, we can sort the bounding boxes of the polytopes in sub-linear time at each instance (this requires a modified sorting algorithm which sorts only the moving objects). We see very little degradation of frame rate when we add the capability of collision detection to our walkthrough environment.

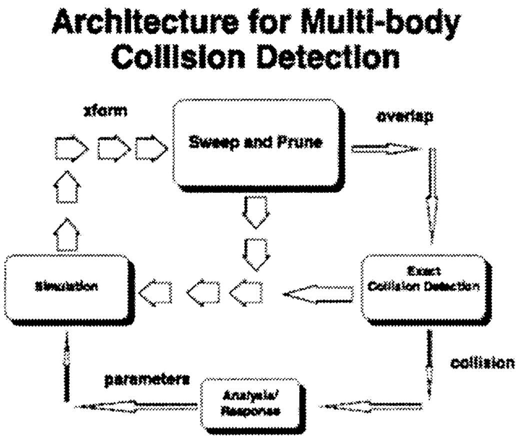


Figure 2: The Architecture for multi-body simulation

We further demonstrate a multi-body simulation with several complex objects in a moderately dense environment. The objects are placed in a bounded 3-dimensional simulation volume whose walls act as barriers. (The architecture of multi-body simulation is shown is Fig. 2.) Unlike the walkthrough environment, *all* the objects move independently. The dynamics of multi-body simulation assumes elastic collision. In order to show the asymptotic time bound on the algorithm, we increase the number of possible interactions quadratically, but the frame time using our collision detection algorithm increases only linearly.

- **Acknowledgements** We are grateful to Dr. Fred Brooks and the walkthrough group at UNC Chapel Hill for the model of the kitchen.

References

[1] J. Cohen, M. Lin, D. Manocha, and K. Ponamgi. Interactive and exact collision detection for large-scaled environments. Technical Report TR94-005, Department of Computer Science, University of North Carolina, 1994.

[2] M. C. Lin. *Efficient Collision Detection for Animation and Robotics.* PhD thesis, University of California at Berkeley, December 1993. Department of Electrical Engineering and Computer Science.

Almost Optimal Polyhedral Separators

HERVÉ BRÖNNIMANN*

Department of Computer Science
Princeton University
Princeton, NJ 08544, USA
E-mail: hbr@cs.princeton.edu

Abstract

We animate two deterministic polynomial time methods for finding a separator for two nested convex polyhedra in 3d. While this problem is NP-complete, we show a reduction to set cover. We then animate the greedy method [4] and the weighted method [1].

1 Introduction

Let $Q \subseteq P$ be two convex polyhedra in $\mathbb{R}^3$. The problem of finding a separating convex polyhedra between P and Q with as few facets as possible is believed to be $\mathcal{NP}$-hard [2], but one can find good approximations for it. Mitchell and Suri [4] cast the problem as a set cover problem. Let $\mathcal{H} = \mathcal{H}(Q)$ denote the set of supporting hyperplanes of Q. Let the *cover set system* be $(\mathcal{H}, \{\mathcal{H}_p : p \in \partial P\})$, where, for any point p, $\mathcal{H}_p$ consists of those hyperplanes in $\mathcal{H}$ for which p and Q lie on opposite sides. They show that a hitting set for the cover set system gives a subset of facets of Q whose supporting halfspaces define a polytope whose boundary lies between Q and P.

A polytope Q' between Q and P such that each facet of Q' is supported by a facet of Q is called *canonical*. The smallest (with respect to the number of facets) canonical polytope contained in P can be obtained from a minimal hitting set of this set system and is within 3 times the optimal separating polyhedron (in the number of faces). Therefore, in this video, we just show how to obtain an optimal canonical separator.

2 Contents of the video

We first familiarize the viewer with the two polyhedra $Q \subseteq P$; Q is seen inside by means of transparency (Fig. 1). We explain how to obtain canonical separators, and how to construct the set system. The portion of the arrangement $\mathcal{A}$ of $\mathcal{H}(Q)$ on the boundary of P is vizualized (Fig. 2), and some rows of the set system are then shown sequentially (Fig. 3), one corresponding to each facet of this arrangement.

To explain the greedy method on this set system, an approach which is proposed by Mitchell and Suri [4], we show how the choice of a facet removes a portion of the boundary of P (Fig. 4), and how the greedy method proceeds sequentially by choosing the facet which covers the biggest remaining portion (measured by the number of cells of $\mathcal{A}$ it separates from Q). After four iterations, we skip to the final greedy separator, which has 14 facets. In general, the greedy separator is guaranteed to have at most $s_{opt} \log |Q|$ facets, where s_{opt} is the size of an optimal canonical separator.

Another algorithm is then animated. It is the weighting method described by Goodrich and the author in [1]. We show how putting a weight on the faces of Q induces a weight for the cells of the arrangement $\mathcal{A}$, and how keeping only the heavy cells yields a separator which is nothing else than a $(1/2c)$-net for the planes $\mathcal{H}(Q)$ supporting faces of Q (Fig. 5). We then explain the reweighting strategy in this context. Eventually, a separator is found that has 13 facets (Fig. 6). In general, the weighted separator is guaranteed to have $O(s_{opt})$ facets.

3 Implementation notes

Both algorithms are implemented and animated using GASP, a Geometric Animation System created at Princeton University by Ayellet Tal and David Dobkin [3]. The implementation is written in C and runs on a Silicon Graphics Iris workstation. The algorithm is fully animated, in the sense that it produces an animation for any given data P and Q. The data used for the video were created with a random generator of

*Supported in part by NSF Grant CCR-93-01254, the Geometry Center, University of Minnesota, an STC funded by NSF, DOE, and Minnesota Technology Inc., and Ecole Normale Supérieure of Paris.

planes tangent to the sphere, then intersected to obtain Q, which was blown up to yield P.

Acknowledgements

We would like to thank Ayellet Tal and David Dobkin for the customized GASP environment and their invaluable help. Thanks also go to Kirk Alexander and Kevin Perry, of Interactive Computer Graphics Lab at Princeton University and Mike Mills, of the Princeton Department of Media Services.

References

[1] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. These proceedings.

[2] G. Das. *Approximation schemes in computational geometry.* Ph.D. thesis, University of Wisconsin, 1990.

[3] D. P. Dobkin and A. Tal. Gasp–a system to facilitate animating geometric algorithms (Video). These proceedings.

[4] J. S. B. Mitchell and S. Suri. Separation and approximation of polyhedral surfaces. In *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, pages 296–306, 1992.

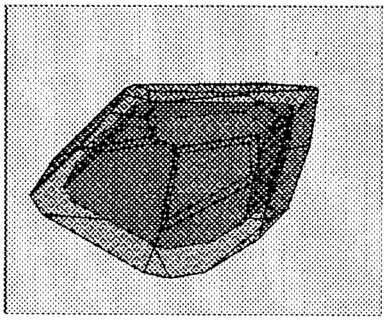


Figure 1: Shown here are the two nested polyhedra.

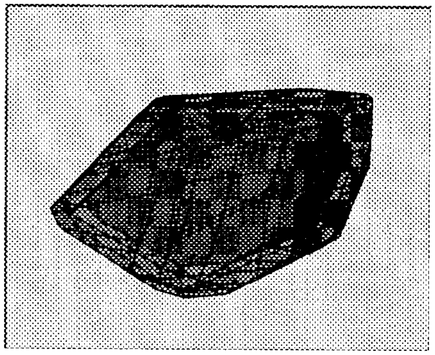


Figure 2: The arrangement of $\mathcal{H}(Q)$ on the boundary of P. Each cell generates a row of the cover set system.

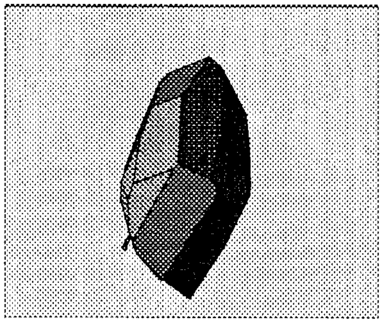


Figure 3: The row generated by a particular cell consists of the facets of Q visible from that cell (highlighted in the picture).

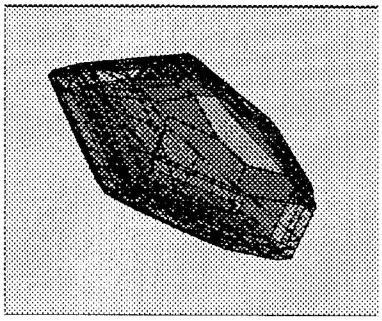


Figure 4: The greedy method chose the highlighted facet on Q, thereby chopping off a few cells on P.

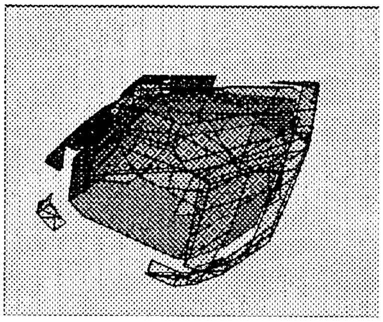


Figure 5: The set of heavy cells on P. The cover for those cells is a $(1/2c)$-net for the planes with weight. It is shown highlighted on th boundary of Q.

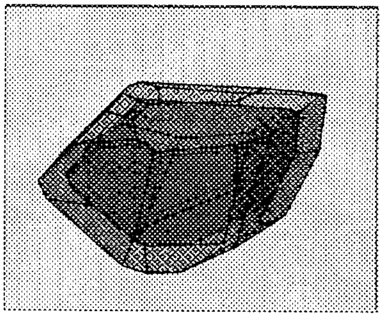


Figure 6: The weighted separator is shown in transparency between P and Q.

Interactive Visualization of Weighted Three-dimensional Alpha Hulls

Amitabh Varshney Frederick P. Brooks, Jr. William V. Wright

Department of Computer Science
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599-3175

Abstract

An interactive visualization of weighted three-dimensional α-hulls is presented for static and dynamic spheres. The α-hull is analytically computed and represented by a triangulated mesh. The entire surface is computed and displayed in real-time at interactive rates. The weighted three-dimensional α-hulls are equivalent to smooth molecular surfaces of biochemistry. Biochemistry applications of interactive computation and display of α-hulls or smooth molecular surfaces are outlined.

1 Introduction

The α-hull has been defined as a generalization of the convex hull of point-sets by Edelsbrunner, Kirkpatrick, and Seidel [4, 3]. Given a set of points P, a ball b of radius α is defined as an *empty α-ball* if $b \cap P = \phi$. For $0 \leq \alpha \leq \infty$, the *$\alpha$-hull* of P is defined as the complement of the union of all empty α-balls [3]. It has been shown that it is possible to compute the α-hull of a set of points P from the Voronoi diagram of P. For $\alpha = \infty$ the α-hull over the set of points P is the same as their convex hull. Edelsbrunner and Mücke [5] have defined the *α-shape* over P to be the polytope that approximates the α-hull over P by replacing circular arcs of the α-hull by straight edges and spherical caps by triangles. An α-shape of a set of points P is a subset of the Delaunay triangulation of P. Edelsbrunner [3], has extended the concept of α-shapes to deal with weighted points (i.e. spheres with possibly unequal radii) in three dimensions. An α-shape of a set of weighted points P_w is a subset of the regular triangulation of P_w.

The smooth molecular surface of a molecule is defined as the surface which an exterior probe-sphere touches as it is rolled over the spherical atoms of that molecule. This definition of a molecular surface was first proposed by Richards [8]. This surface is useful in studying the structure and interactions of proteins, in particular for attacking the protein-substrate docking problem. The analytic computation of the molecular surface was first done by Connolly [2].

Looking at the above definitions, one can see that the weighted α-hulls for three dimensions and the smooth molecular surface of a molecule with a probe-radius α have the same definitions. In this video we present the visualization of weighted α-hulls as used to define molecular surfaces.

2 Our Approach

Computation of α-hulls and α-shapes has traditionally been done by first constructing power diagrams or regular triangulations. Since these methods involve computing the entire diagram or triangulation first and then culling away the parts that are not required, their complexity is $O(n^2)$ in time, where n is the number of points. This is worst-case optimal, since an α-shape in three dimensions could have a complexity of $\Omega(n^2)$.

However, when α-hulls are used as molecular surfaces, one can do better. Let $M = \{S_1, \ldots, S_n\}$, be a set of spheres, where each sphere, S_i, is expressed as a pair (c_i, r_i), c_i being the center of the sphere and r_i being the radius of the sphere. Collections of spheres representing molecules have two interesting properties: (i) the minimum distance d_{ij} between any two centers c_i and c_j is greater than or equal to a positive constant l_{min} — the smallest bond-length in the molecule and (ii) the values of all the radii can be bounded from above and below by strictly positive values, $0 < r_{min} \leq r_i \leq r_{max}$. We take advantage of the first property to arrive at better running times for our algorithm. Stated simply, the first property says that the number of neighboring atoms within a fixed distance from any atom i, is always bounded from above by a constant k_{max} that depends on the minimum spacing between any two atoms. We refer to two atoms i and j as *neighboring* if it is possible to place a probe sphere such that it is contact with both S_i and S_j.

A power cell (as described by Aurenhammer [1]) for a given sphere S_i with respect to M can be computed as the intersection of $n-1$ halfspaces, each halfspace contributed by the pair $(S_i, S_j), 1 \leq j \leq n, j \neq i$. If the average number of neighbors for an atom is k, then for our purposes it is sufficient to just compute an approximation to the power cell, which we call a *feasible cell*, as an intersection of k halfspaces (one halfspace contributed by each neighbor). This can be done in deterministic time O($k \log k$). For n atoms, this task can be parallelized over n processors, each processor computing the feasible cell for one atom. To check if a feasible cell is non-null, we use a randomized linear programming algorithm that has linear expected time complexity and is quite fast in practice [9]. For details of our approach the interested reader can refer to [10].

We have used ideas from the theory of packing and covering of spheres to estimate the value for k, the average number of neighboring atoms, for an atom in protein molecules. We can prove that for proteins, $k < 140$ for a probe-sphere radius of $1.4\AA$ – the radius of a water molecule. Details about this result can be found in [11]. In practice we have found that for protein molecules and for a probe-radius of $1.4\AA$, k is around 45

In the algorithms for computing the convex hull of a set of points, it is assumed that the points are in a general position, i.e. no more than d points lie on the same $d-1$ dimensional hyperplane. In reality this assumption often fails to hold, leading to problems. For example, planar benzene rings occur often in proteins, causing six carbon and six hydrogen atoms to be all coplanar. One of the recent approaches to solving this problem has been to perturb the input point set slightly to avoid these degeneracies. The approach of Emiris and Canny [6] perturbs the j^{th} dimension of the i^{th} point as:

$$p_{i,j}(\epsilon) = p_{i,j} + \epsilon(i^j \bmod q) 1 \leq i \leq n, 1 \leq j \leq d \qquad (1)$$

where ϵ is a symbolic infinitesimal and q is the smallest prime greater than n. Instead of performing exact integer arithmetic, we just perturb the centers of the spheres by the above scheme and that has worked quite well for us in practice.

With no preprocessing, we can compute the molecular surface for a 396 atom Crambin and a probe-radius of $1.4\AA$, using 40 Intel i860 processors in 0.2 seconds. We have implemented α-hulls on Pixel-Planes 5 [7], though the method is general enough to be easily implemented on any other parallel architecture. Our approach is analytically exact. The only approximation is the degree of tessellation of spherical and toroidal patches.

3 Applications in Biochemistry

Interactive computation and display of molecular surfaces should benefit biochemists in three important ways. First, the ability to change the probe-radius interactively helps one study the surface. Second, it helps in visualizing the changing surface of a molecule as its atom positions are changed. These changes in atom positions could be due to user-defined forces as the user attempts to modify a molecular model on a computer. Third, it assists in incorporating the effects of the solvent into the overall potential energy computations during the interactive modifications of a molecule on a computer.

4 Scope for further work

At present we are not using any incremental temporal information in constructing these surfaces. Thus, if the atoms move slightly from their positions, the whole surface is recomputed from the beginning, although one could conceivably store the feasible cell description with each atom. Assuming the atoms of the molecule move along continuous trajectories, it should be possible to compute such surfaces (and indeed α-hulls and α-shapes) incrementally and efficiently by using the information from previous time steps.

Acknowledgements

We would like to acknowledge the valuable discussions we had with Pankaj K. Agarwal, Dinesh Manocha, Jan F. Prins, and David C. Richardson during various stages of this work. We would also like to acknowledge Michael E. Pique and Victoria Roberts at the Scripps Clinic for the molecular dynamics data. We would like to thank the anonymous referees for their suggestions which have led to improvements in the presentation of this extended abstract as well as the video. This work was supported by NIH National Center for Research Resources grant number 5-P41-RR02170.

References

[1] F. Aurenhammer. Power diagrams: Properties, algorithms and applications. *SIAM Journal of Computing*, 16(1):78–96, 1987.

[2] M. L. Connolly. Analytical molecular surface calculation. *Journal of Applied Crystallography*, 16:548–558, 1983.

[3] H. Edelsbrunner. Weighted alpha shapes. Technical Report UIUCDCS-R-92-1740, Department of Computer Science, University of Illinois at Urbana-Champaign, 1992.

[4] H. Edelsbrunner, D. G. Kirkpatrick, and R. Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, IT-29(4):551–559, 1983.

[5] H. Edelsbrunner and E. P. M"ucke. Three-dimensional alpha shapes. *ACM Transactions on Graphics*, 13(1), 1994.

[6] I. Emiris and J. Canny. An efficient approach to removing geometric degeneracies. In *Eighth Annual Symposium on Computational Geometry*, pages 74–82, Berlin, Germany, June 1992. ACM Press.

[7] H. Fuchs, J. Poulton, J. Eyles, T. Greer, J. Goldfeather, D. Ellsworth, S. Molnar, G. Turk, B. Tebbs, and L. Israel. Pixel-planes 5: A heterogeneous multiprocessor graphics system using processor-enhanced memories. In *Computer Graphics: Proceedings of SIGGRAPH'89*, volume 23, No. 3, pages 79–88. ACM SIGGRAPH, 1989.

[8] F. M. Richards. Areas, volumes, packing and protein structure. *Ann. Rev. Biophys. Bioengg.*, 6:151–176, 1977.

[9] R. Seidel. Linear programming and convex hulls made easy. In *Sixth Annual ACM Symposium on Computational Geometry*, pages 211–215, Berkeley, California, June 1990. ACM Press.

[10] A. Varshney and F. P. Brooks, Jr. Fast analytical computation of Richards's smooth molecular surface. In G. M. Nielson and D. Bergeron, editors, *IEEE Visualization '93 Proceedings*, pages 300–307, October 1993.

[11] A. Varshney, W. V. Wright, and F. P. Brooks, Jr. Bounding the number of unit spheres inside a larger sphere. Technical Report UNC-CS-TR-93-039, Department of Computer Science, University of North Carolina at Chapel Hill, 1993.

Index